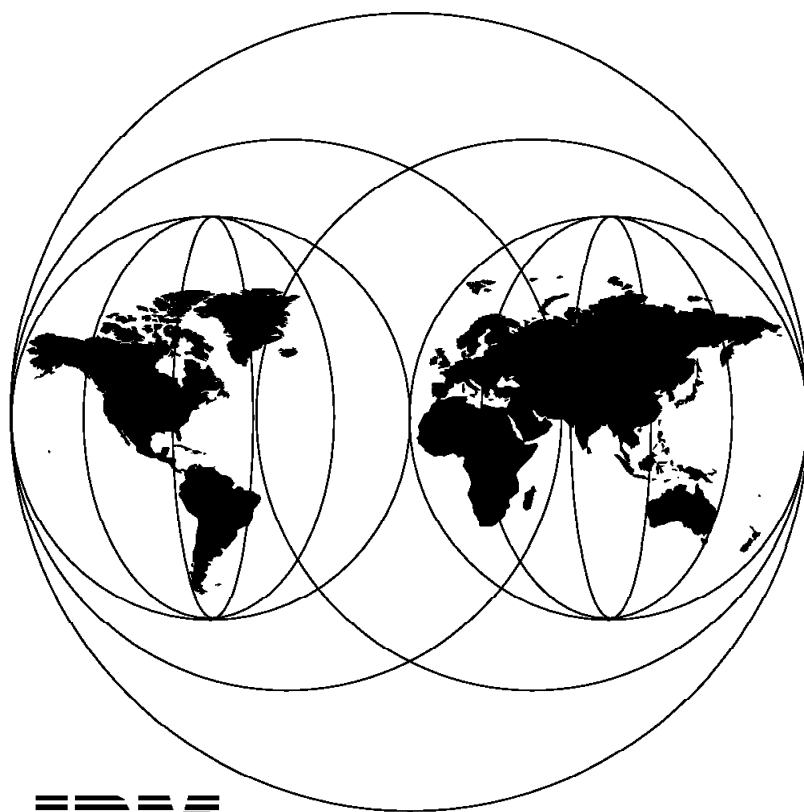


Migrating Micro Focus OS/2 Dialog System Applications to IBM VisualAge for COBOL for OS/2

December 1996



IBM

**International Technical Support Organization
San Jose Center**



International Technical Support Organization

SG24-4887-00

**Migrating Micro Focus OS/2 Dialog System Applications
to IBM VisualAge for COBOL for OS/2**

December 1996

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix A, "Special Notices" on page 59.

First Edition (December 1996)

This edition applies to Version 1.2 Refresh, 83H9131, VisualAge for COBOL for OS/2 for use with the OS/2 WARP 3.0 Operating System.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. QXXE Building 80-E2
650 Harry Road
San Jose, California 95120-6099

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© **Copyright International Business Machines Corporation 1996. All rights reserved.**

Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

Figures	v
Preface	vii
How This Redbook Is Organized	vii
The Author of This Redbook	viii
Comments Welcome	viii
Chapter 1. Introduction	1
1.1 Description of a Typical GUI MF COBOL Application	1
1.2 Description of a Typical IBM COBOL PM Application	3
1.3 The Migration Approach	5
1.3.1 Changing from DS to VDE	6
1.3.2 Translating from the Dialog System Code to the IBM COBOL Structure	6
1.3.3 Moving from MF COBOL to IBM COBOL	7
1.3.4 Data Migration	8
Chapter 2. Sample Application Description	11
2.1 Before You Start	11
2.2 Our Approach	11
2.3 The Application	11
Chapter 3. Migration Utility Overview	17
3.1 Background	17
3.2 Utility Description	17
3.3 How to Use the Migration Utility	21
Chapter 4. Comparison of MF Dialog System Objects and VisualAge COBOL	
Parts	25
4.1 General Differences	25
4.2 Windows	25
4.3 Menu Items	26
4.4 Entry Fields	27
4.5 Push Buttons	29
4.6 List Boxes	30
4.7 Multiline Entry Fields	31
4.8 Radio Button	31
4.9 Selection or Combination Boxes	31
4.10 Static Text	32
4.11 Group Boxes	32
4.12 Bitmaps	32
4.13 Notebooks	32
4.14 Dialog Boxes	32
4.15 Message Boxes	33
Chapter 5. Migrating the Sample Application	35
5.1 A Practical Example	35
Chapter 6. Embedding COBOL Logic in the Sample Application	39
6.1 Embedding the Old Code (the OLDMAIN procedure)	42

Chapter 7. Event Logic Migration to VisualAge COBOL for Sample Application	45
7.1 Event to Event Logic	45
7.1.1 Events for the OEMAIN-CUSTNO Entry Field	48
7.1.2 Events for the CUSTINFO-SHIPID Entry Field	51
7.1.3 Events for the OEMAIN-PARTNO Entry Field	51
7.1.4 Events for the OEMAIN-PRODUC-MODEL Entry Field	52
7.1.5 Events for the OEMAIN-UNIT Entry Field	52
7.1.6 Events for the OEMAIN-CUSTINFO Push Button	53
7.1.7 Events for the ORDEROK Radio Button	54
7.1.8 Events for the OEMAIN-NEWORDER Push Button	55
7.1.9 Events for the REVISE-ORDER Radio Button	55
Chapter 8. Installing the Migration Utility	57
8.1 Hardware and Software Prerequisites	57
8.2 Installing the Migration Utility	57
Appendix A. Special Notices	59
Appendix B. Related Publications	61
B.1 International Technical Support Organization Publications	61
B.2 Redbooks on CD-ROMs	61
B.3 Other Publications	61
How to Get ITSO Redbooks	63
How IBM Employees Can Get ITSO Redbooks	63
How Customers Can Get ITSO Redbooks	64
IBM Redbook Order Form	65
Glossary	67
List of Abbreviations	87
Index	89

Figures

1.	MF DS COBOL Representation	3
2.	IBM VDE Representation	4
3.	MF COBOL versus IBM COBOL	5
4.	Migration Cycle	6
5.	MF COBOL versus IBM COBOL	7
6.	Conversion Requirements for Data File Migration	8
7.	Proposed Migration Tool	9
8.	Main Window of the Celdial Order Entry Application	12
9.	Customer List Window	13
10.	Customer Information Window	14
11.	Select Part Number Window	14
12.	Select Model Window	15
13.	Ship ID Window	15
14.	The Celdial Order Entry	16
15.	Micro Focus DS to IBM VDE Migration Utility Window	18
16.	Sample Micro Focus Dialog System Window	18
17.	Micro Focus DS to IBM VDE Migration Utility Window	21
18.	MF2VDE - Message Window Migration Utility	22
19.	Parts Catalog Window of the Migration Utility	23
20.	Window Properties Window for MF Dialog System	26
21.	Window Part Settings Window for IBM VDE	26
22.	Pull Down Choice Details Window for the MF Dialog System	27
23.	Menu Item Part Settings Window for IBM VDE	27
24.	Entry Field Properties Window for MF Dialog System	29
25.	Entry Field Part Settings Window for IBM VDE	29
26.	Push Button Properties Window for MF Dialog System	30
27.	Push Button Part Settings Window for IBM VDE	30
28.	List Box Properties Window for MF Dialog System	31
29.	List Box Part Settings Window for IBM VDE	31
30.	Message Box Properties Window for MF Dialog System	33
31.	COBOL GUI Designer - Define Message Window for IBM VDE	34
32.	Migration Utility Window	35
33.	Migrated Ship - ID Window Showing Wrong-Sized Push Buttons.	36
34.	VDE Define Message Window Showing Migration Fixes.	38
35.	IBM COBOL Compiler Options Window	39
36.	IBM COBOL Compiler Options Window	40
37.	IBM COBOL Compiler Options Window	41
38.	IBM COBOL Compiler Options Window	42
39.	IBM VDE Code Assistant Window	47
40.	COBOL GUI Designer - Celdial Window	49
41.	MF2VDE - Settings Window Showing the Migration Utility Settings	58

Preface

This redbook describes the process to migrate a Micro Focus Dialog System application on OS/2 to IBM VisualAge for COBOL for OS/2 Version 1.2. The relationship to VisualAge for COBOL for OS/2 Version 2 is that there is a migration capability provided in Version 2 to migrate Version 1.2 Visual Development Environment GUI parts to the Version 2 Visual Builder. The steps for migration and use of a Dialog System migration tool in relationship to Version 1.2 are covered.

This redbook was written for application developers interested in the requirements for migrating Micro Focus Dialog System applications to IBM VisualAge for COBOL for OS/2.

A practical example is used as illustration of the migration process and the Dialog System migration tool is provided.

Knowledge of Micro Focus Dialog System, IBM VisualAge for COBOL for OS/2, the COBOL language, and visual building techniques is required.

How This Redbook Is Organized

This redbook contains 89 pages. It is organized as follows:

- Chapter 1, "Introduction"

Introduces the steps required to migrate a Micro Focus Dialog System application to VisualAge for COBOL.

- Chapter 2, "Sample Application Description"

This chapter gives an overview of the sample application.

- Chapter 3, "Migration Utility Overview"

Details on the Migration Utility are reviewed in this chapter.

- Chapter 4, "Comparison of MF Dialog System Objects and VisualAge COBOL Parts"

This chapter compares various Dialog System objects to VisualAge COBOL parts and explains the differences when migrating an application.

- Chapter 5, "Migrating the Sample Application"

The steps used to migrate the sample application are reviewed.

- Chapter 6, "Embedding COBOL Logic in the Sample Application"

This chapter explains porting the COBOL business logic to the migrated application.

- Chapter 7, "Event Logic Migration to VisualAge COBOL for Sample Application"

The steps that need to be executed to set the actions for each part are explained.

- Chapter 8, "Installing the Migration Utility"

Steps to unpack and install the Migration Utility are explained.

The Author of This Redbook

This redbook was produced by:

Agostino Assi, IBM Italy.

Thanks to the following people for their invaluable contributions to this project:

Paul Pacholski
IBM Toronto

Wilbert Kho
IBM Santa Teresa Laboratory, San Jose, California

This project was designed and managed by:

Joe DeCarlo
IBM International Technical Support Organization, San Jose Center

Comments Welcome

Your comments are important to us!

We want our redbooks to be as helpful as possible. Please send us your comments about this or other redbooks in one of the following ways:

- Use the electronic evaluation form found on the Redbooks Home Pages at the following URLs:

For Internet users <http://www.redbooks.ibm.com>

For IBM Intranet users <http://w3.itso.ibm.com/redbooks>

- Send us a note at the following address:

redbook@vnet.ibm.com

Chapter 1. Introduction

Often when we discuss migrations, we mean moving from one version of a product to a later version. Typical examples are migrations from OS/VS COBOL to VS COBOL II or from VS COBOL II to COBOL/370.

In this type of migration we most often stay with the same product, merely changing syntactical structures or introducing new functions.

Another type of migration, less frequent perhaps because it is more complex, requires a change of product (moving from one tool to another). In this type of migration many factors must be addressed, such as:

- Language to language migration
- Data format migration
- Dialog format migration
- Different approach to handling events.

These four points make the migration from the Micro Focus (MF) Dialog System (DS) application development environment to the IBM VisualAge for COBOL for OS/2 more complex than the standard language-to-language migration.

Before starting on the technical details that characterize this type of migration, we briefly describe how both tools are characterized and how best to approach the migration.

Hereafter, IBM VisualAge for COBOL for OS/2 can be referred to as VisualAge COBOL, VA COBOL, Visual Development Environment (VDE), or Graphical User Interface (GUI) Designer.

1.1 Description of a Typical GUI MF COBOL Application

A Typical MF GUI COBOL application consists of:

- A Graphical System (the .GS file you create using the MF Dialog System tool)
- A .CPB file that contains all the declarations you need for the variables you defined in the Dialog System. It is a sort of bridge between the DS and COBOL.
- A .CBL program (the heart of the application):

```
....
working-storage section.
    copy "ds-cntrl.mf".
    copy "celdial.cpb".
    copy "sqlenv".
    copy ".....".
01 xxxx    .....

procedure division.
main-process section.
    perform program-initialize.
    perform program-body until exit-pgm-true.
    perform program-terminate.
```

```

program-initialize section.
.....
program-body section.
  evaluate true
    when custlist-botton=true
      perform show-customer-list
    when .....-true
      perform .....
  end-evaluate
  perform call-dialog-system.

program-terminate section.
.....
  stop run.

load-screenset section.
  move ds-new-set to ds-control
  move "c:\dsg\celdial\celdial.gs" to ds-set-name
  perform call-dialog-system.

call-dialog-system section.
  call dialog-system using ds-control-block,
                        data-block.
  if not ds-no-error
    move ds-error-code to display-error-no
    display "DS ERROR NO: " display-error-no
    perform program-terminate
  end-if.
.....

```

In the MF environment, the business logic is inside the program, while the logic needed to manage the GUI is in the .DS file. To implement the logic inside the .DS file, MF unfortunately uses another language (not COBOL) which needs a translation and conversion to IBM's Visual Development Environment (VDE) which is used in IBM VisualAge for COBOL for OS/2. This is one of the differences we will meet during our migration and it will also be one of our major problems.

A graphical representation of how components talk to one another in the MF environment is shown in Figure 1.

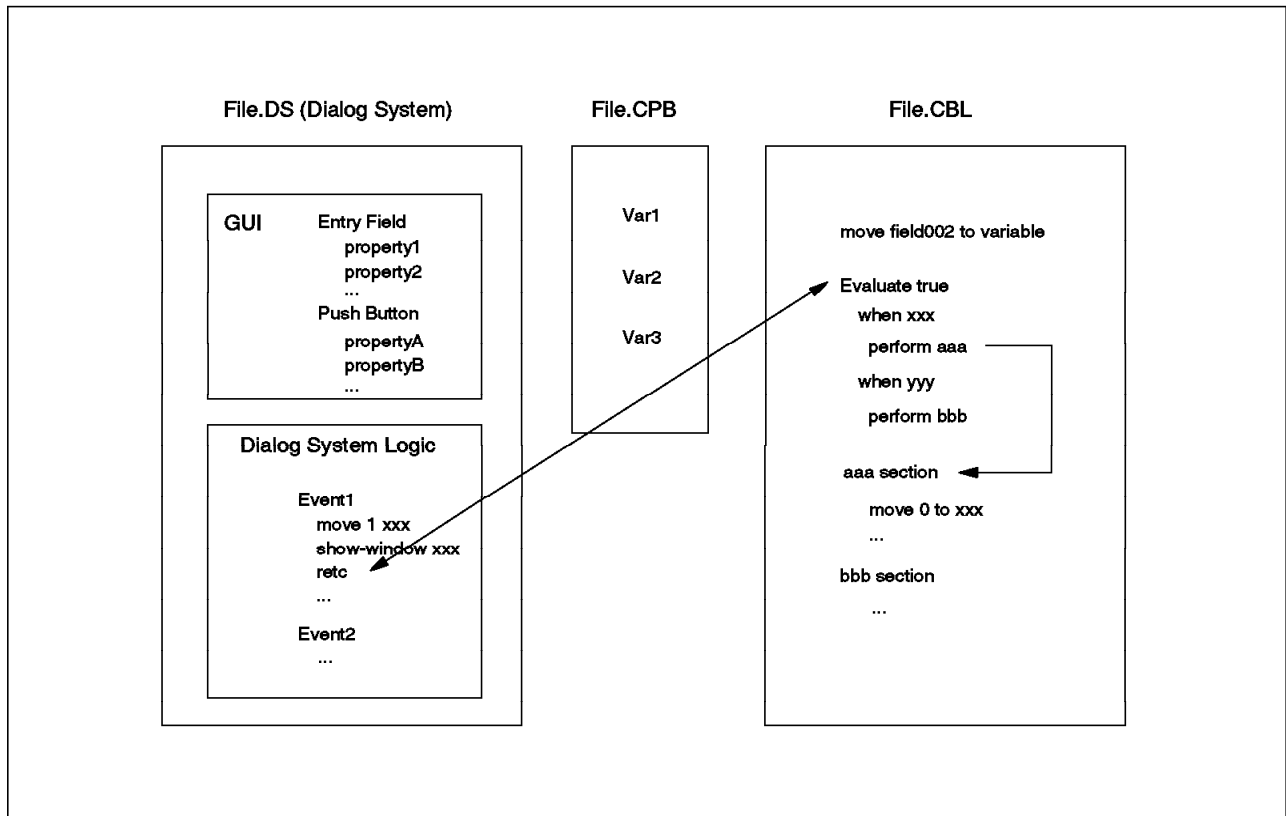


Figure 1. MF DS COBOL Representation. Graphical representation of the structure of a Micro Focus application.

When you execute an event, you execute a piece of code written in the DS language which, at a certain moment, passes the control to a COBOL program. The COBOL program passes the control back to the Dialog System, which in turn executes a portion of its code and then passes the control back to the Presentation Manager (PM), which is waiting for another event.

Values and variables used in the COBOL program can be used to update entry fields, list boxes, or other dialog objects. In order to automatically update COBOL variables after refreshing the window which they belong to, MF links a dialog system object with one or more COBOL variables.

1.2 Description of a Typical IBM COBOL PM Application

A typical IBM COBOL Presentation Manager application is composed of a GUI driver, which handles the user interface (UI) dialog and a COBOL program that contains the logic you need to control and handle the UI dialog and the business processes. When using the IBM solution, you need not know two languages; COBOL is the only language necessary to write a PM application (Figure 2). This means that part of the migration is a code translation of sorts between the DS language and IBM COBOL.

Another characteristic that makes the tools different is that IBM COBOL uses *entires* to define events, while MF uses *procedures* and calls them by setting a flag and performing the related procedure in an Evaluate Loop.

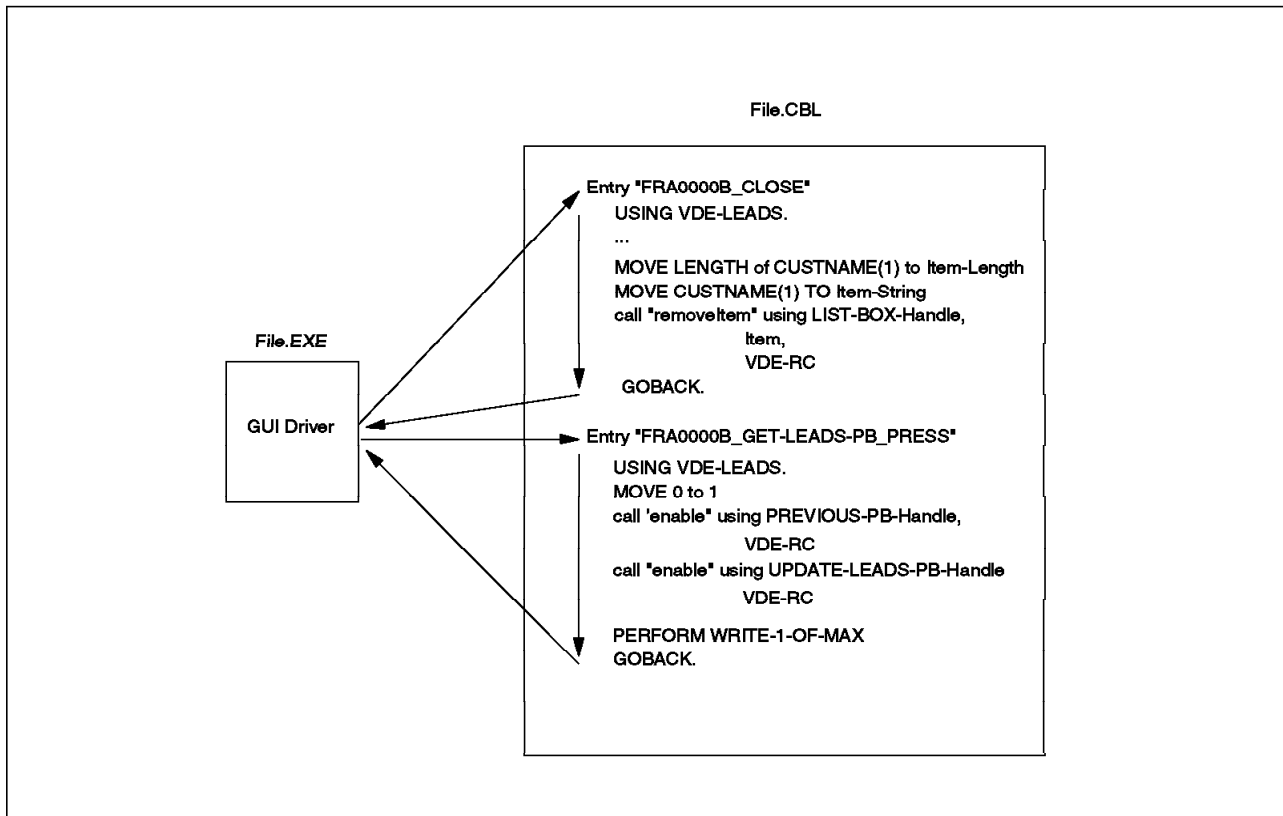


Figure 2. IBM VDE Representation. Graphical representation of the structure of an IBM COBOL VisualAge application.

Because the MF Dialog System uses two different languages, in the starting environment you must define variables related to some object (entry fields, list boxes, multiline edit, and the like) in order to pass and get back values to the COBOL program which is processing the logic. Therefore, between the COBOL logic and the Dialog System logic, there exists a bridge based on these variables which works like an interface service (Figure 3).

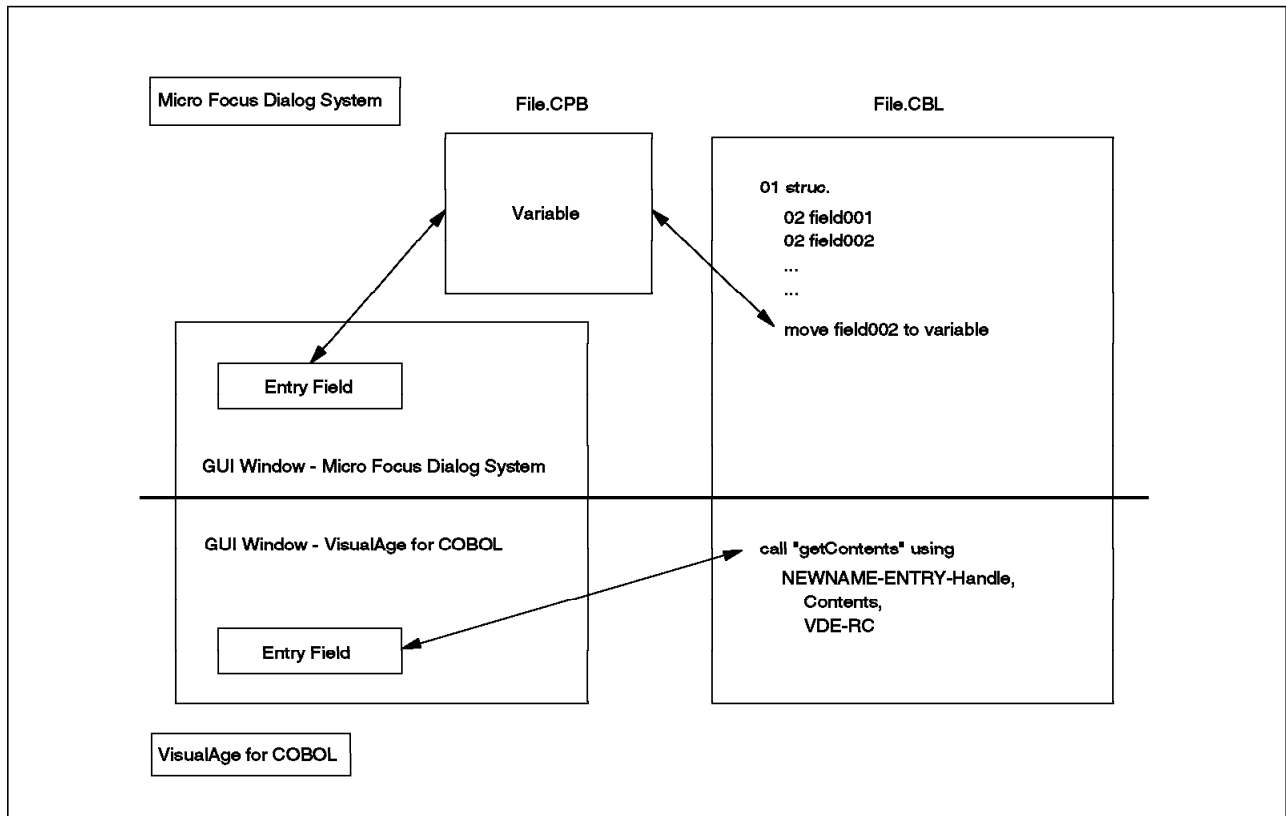


Figure 3. MF COBOL versus IBM COBOL. How behavior differs between the Micro Focus Dialog System and IBM VDE when we update an entry field.

1.3 The Migration Approach

The approach used is divided into four steps:

1. Go from DS to VDE.
2. Convert the Dialog System code to the IBM COBOL structure.
3. Go from MF COBOL to IBM COBOL.
4. Migrate the data.

Figure 4 is a graphical representation of the desired migration method.

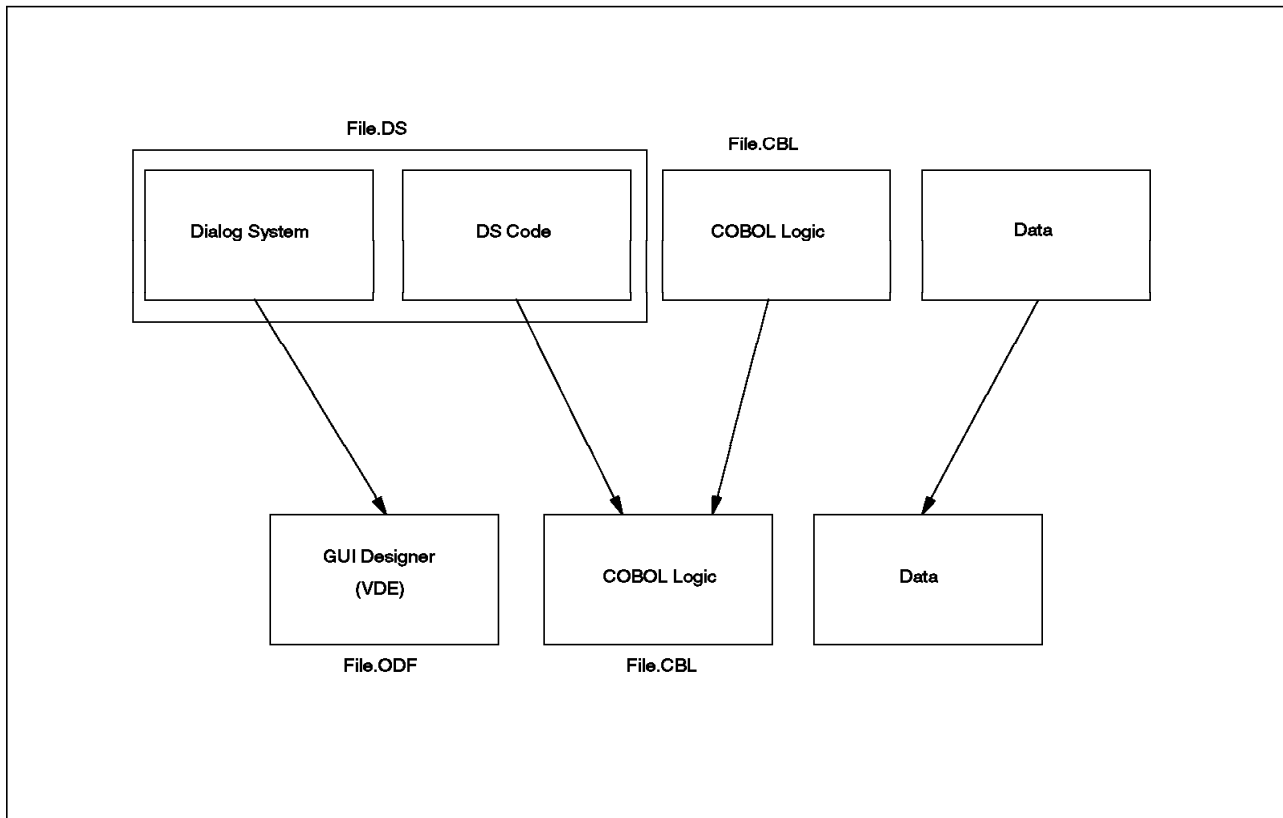


Figure 4. Migration Cycle

1.3.1 Changing from DS to VDE

This step normally involves a detailed analysis of the entire application being migrated: first looking for the related objects in the VDE, then creating them manually, and setting their properties one by one.

Instead, we wrote a Migration Utility procedure that assists in creating some of these objects with their related properties. We describe how to use the Migration Utility in Chapter 3. For now, we assume that part of this first step can be performed automatically. Figure 5 summarizes the steps.

1.3.2 Translating from the Dialog System Code to the IBM COBOL Structure

This step must be performed manually. The differences between the two tools greatly affect this particular part of the migration. We had to use two machines simultaneously and more than one open editor for each machine in order to determine the flow of the MF environment and translate in into the IBM COBOL code.

From MF Dialog System to VisualAge for COBOL

- > Object by object you have to recreate the whole layout
(not all the object properties are supported)
- > For example, Message boxes in VisualAge COBOL are implemented differently

From DS language to VisualAge for COBOL

- > Event by event we have to translate the Dialog System code into VisualAge COBOL code

From MF COBOL to VisualAge for COBOL

- > You need only to copy and paste the business logic from the MF source code into the VisualAge COBOL source code

From MF data format to IBM data format

- > VSAM and sequential file formats are different; you need to write some conversion programs

Figure 5. MF COBOL versus IBM COBOL. Summary of steps to perform to migrate from MF to IBM COBOL.

1.3.3 Moving from MF COBOL to IBM COBOL

The sample application we used did not give us the opportunity to find any particular aspect of COBOL differences worth comment. Within the boundaries of the American National Standards Institute/ the ANSI 85 standard, no problem should occur.

1.3.4 Data Migration

Since the only difference we have found is in the format of VSAM and sequential files, we suggest writing a small program that reads data from the MF format and writes into a flat file. Then, with another program, you can read that data and recreate the files in the IBM COBOL format (Figure 6).

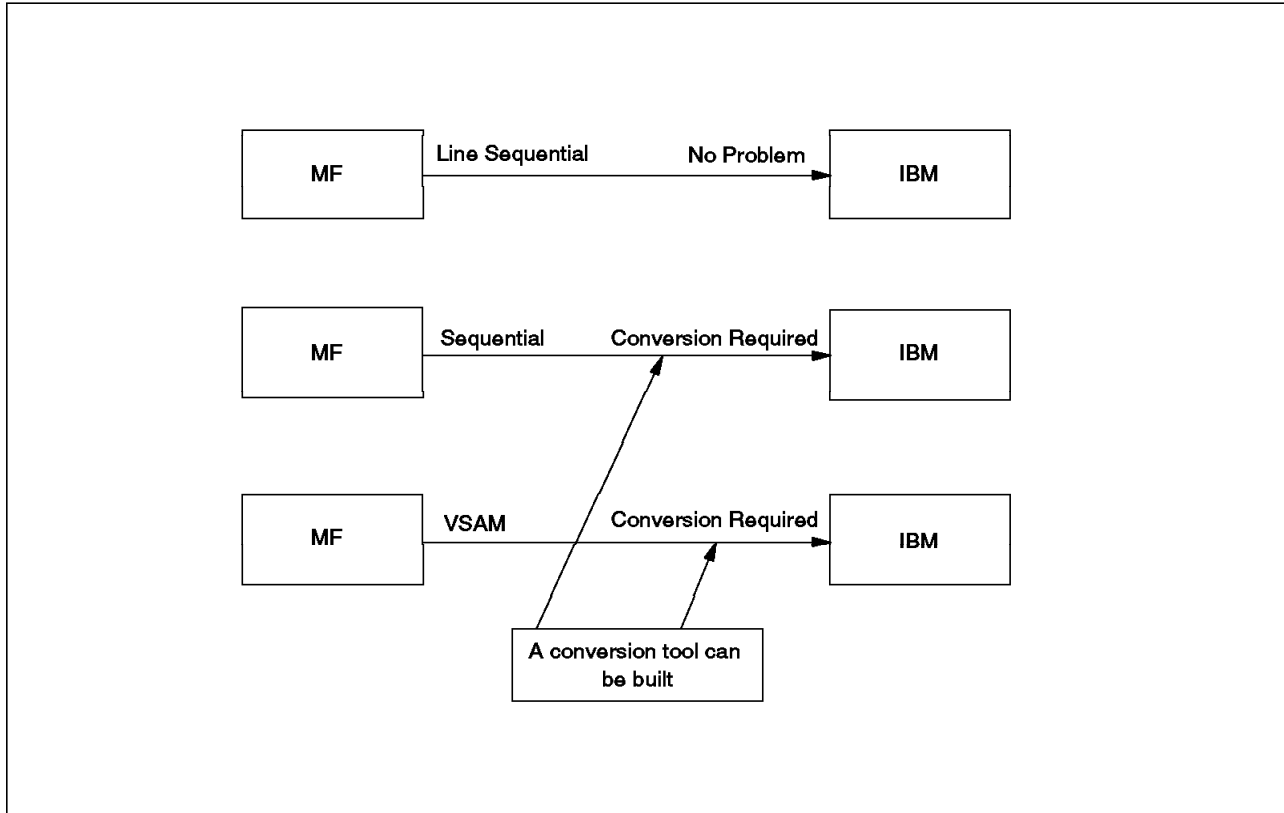


Figure 6. Conversion Requirements for Data File Migration

We have designed for a tool to automate the migration and that tool will be covered in another book which focuses on data migration only. Figure 7 shows how that tool behaves.

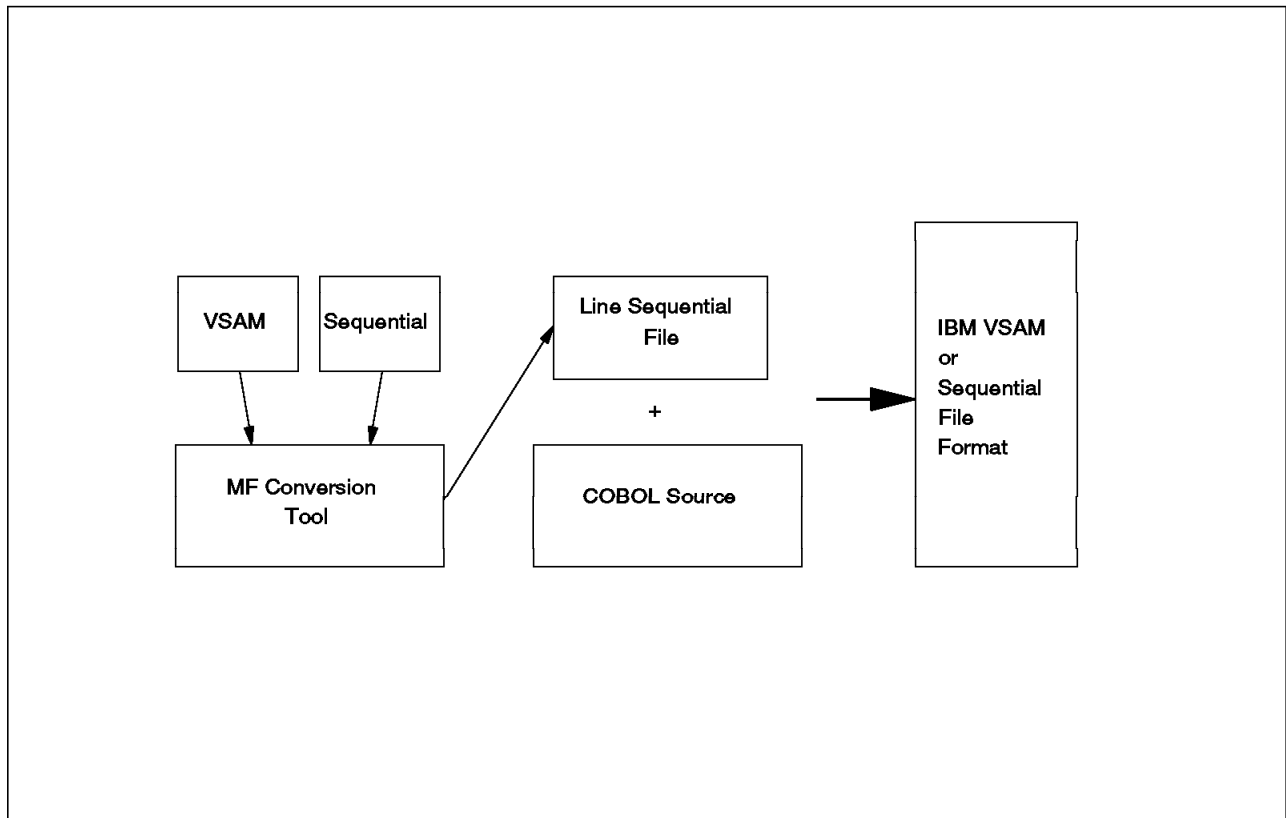


Figure 7. Proposed Migration Tool. Flow for the routine that would have migrated VSAM and sequential files automatically.

Chapter 2. Sample Application Description

2.1 Before You Start

When migrating from one tool to another, you may want to use the same workstation for both. Here are some suggestions and hints to ensure that a coexistence of tools will not affect the migration:

- DB2/2

MF 3.2 supports DB2/2 V.1.x, but does not support DB2/2 V.2.x, which is a prerequisite for IBM VisualAge for COBOL for OS/2. Therefore, you have to have both versions of DB2/2 and be able to switch from one to the other easily.

- LANG environment variable

Both tools use the LANG environment variable. MF does not require the variable to be specified in the CONFIG.SYS file, it uses the default value. IBM COBOL needs the variable to be specified in the CONFIG.SYS file, which changes the value. The changed value is not accepted by MF. The solution is to add the following statement to the CONFIG.SYS file:

```
SET LANG=EN_US
```

This is done to inform MF that the LANG variable is used by another application and its value has been moved to that variable.

- Installation sequence

Depending on which tool you installed first, VisualAge COBOL or DB2/2, you should pay attention to the value of the threads variable in your CONFIG.SYS file. The value should always be greater than 1024.

2.2 Our Approach

The best approach we can use to describe potential migration problems is to use an existing application and explain, step by step, how to migrate it, highlighting critical and delicate steps.

Before beginning with migration suggestions, we briefly describe the application we are going to work with. This application is referred to throughout the document.

2.3 The Application

The application we have chosen is an order entry procedure of the Celdial application. Celdial is a fictitious cellular telephone manufacturer used for demonstration and education purposes.

The Celdial application to be used is a Presentation Manager (PM) application, based on a DB2/2 database, that uses flat files for basic data storage.

The main window of the order entry procedure is shown in Figure 8.

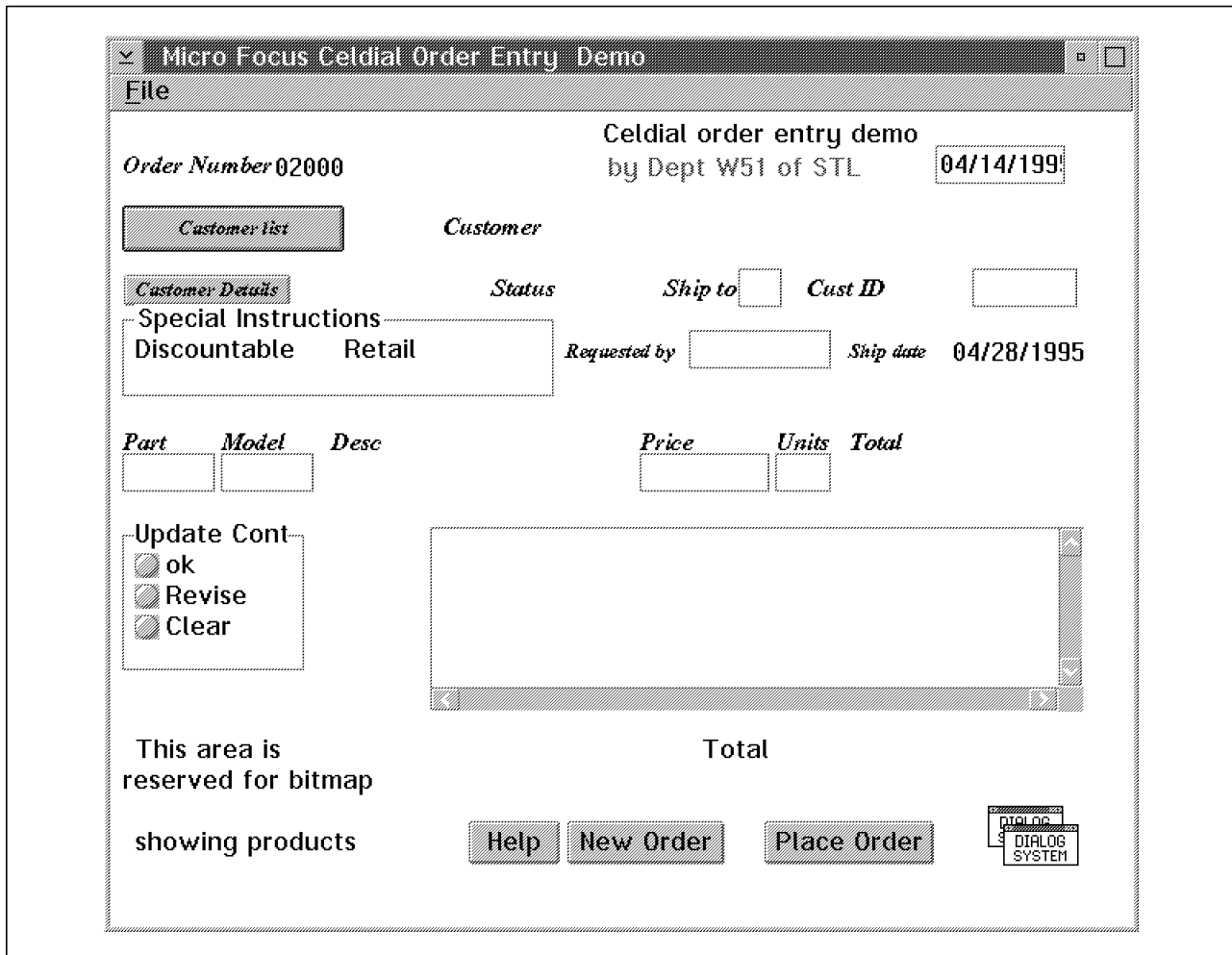


Figure 8. Main Window of the Celdial Order Entry Application

The main window is complex and populated; however, within it you will find several types of objects, such as:

- Entry fields
- Static texts
- Push buttons
- Group boxes
- List boxes
- Radio buttons
- Bitmaps

For many objects properties have been changed. This is a good testing environment for a migration because different kinds of objects and properties have been set and they must be retained in the migrated application. In the main window there are some hidden entry fields, such as the entry field at the right of *Customer* or the entry field below *Desc* and *Total*. During the UI migration, we should be careful that these properties are kept.

We start with a quick description of what the application does and then we look at its windows. The application has been created to help the Celdial clerk handle the product orders. The Celdial database contains business tables, such as:

- Table of customers
- Table of products
- Order Summary table
- Order History table

The entire application is based on this database. The order entry part is composed of six windows:

- Main
- Customer list
- Customer information
- Parts list
- Models list
- Ship ID list

From the main window, press the **Customer List** push button to reach the Customer List window (Figure 9).

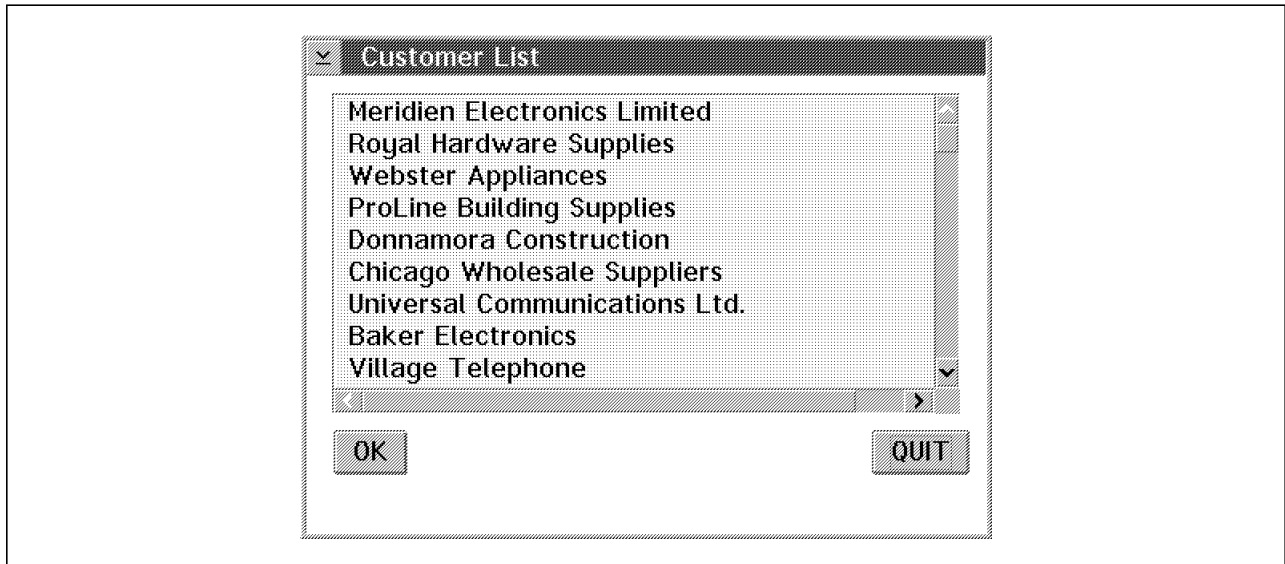


Figure 9. Customer List Window

After choosing a customer from the list, ask for associated information by clicking on **Customer Details**. The Customer Information Window appears (Figure 10).



Customer Information

Customer id Rep ID

Customer

Contact name

Address

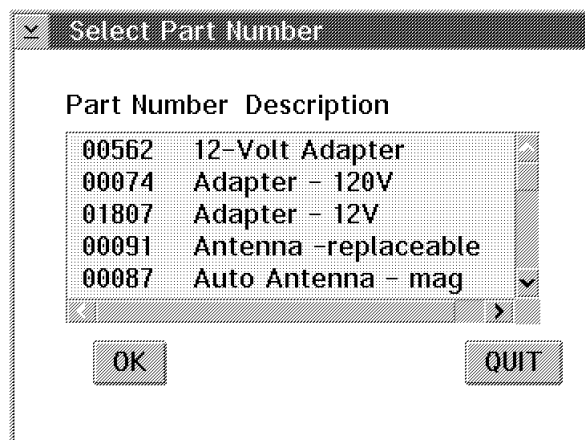
City ZIPLOC

Country Zipcode

Phone Fax Number

Figure 10. Customer Information Window

Once a customer is selected, you can prepare an order for that customer by choosing an available part from the Select Part window To do so, position the cursor on the *part entry* field in the main window and press the PF9 key. The Select Part Number window appears (Figure 11).



Select Part Number

Part Number	Description
00562	12-Volt Adapter
00074	Adapter - 120V
01807	Adapter - 12V
00091	Antenna -replaceable
00087	Auto Antenna - mag

Figure 11. Select Part Number Window

After selecting an item, return to the main window (Figure 8). Position the cursor in the *model entry* field and press the PF9 key to select a model for the part (Figure 12).

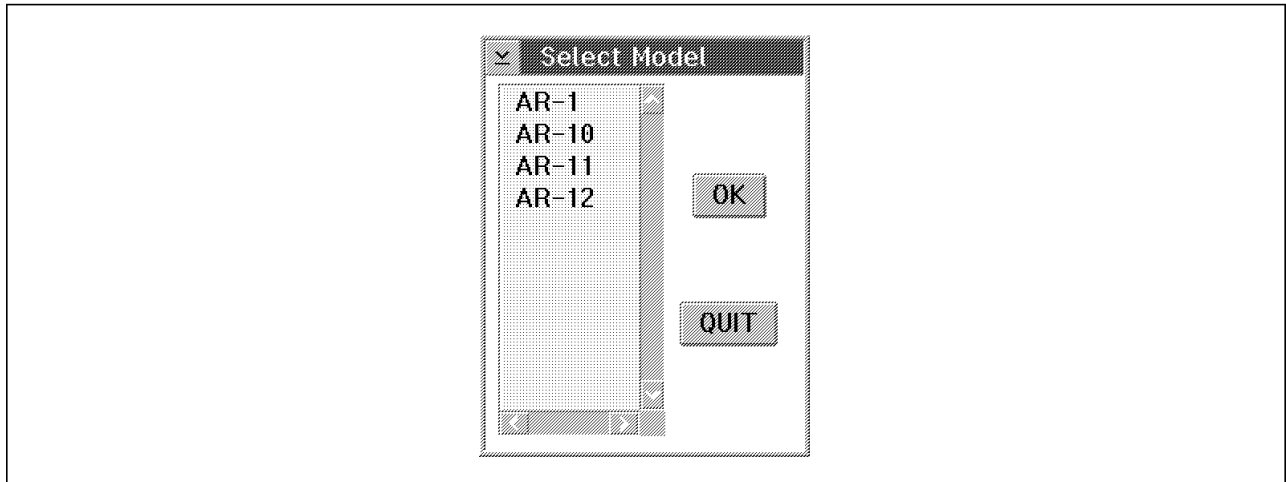


Figure 12. Select Model Window

The lower left corner of the main window in Figure 8 now has a bitmap of the product whose number and model you have selected.

Finally, select the Ship ID for your order (Figure 13).

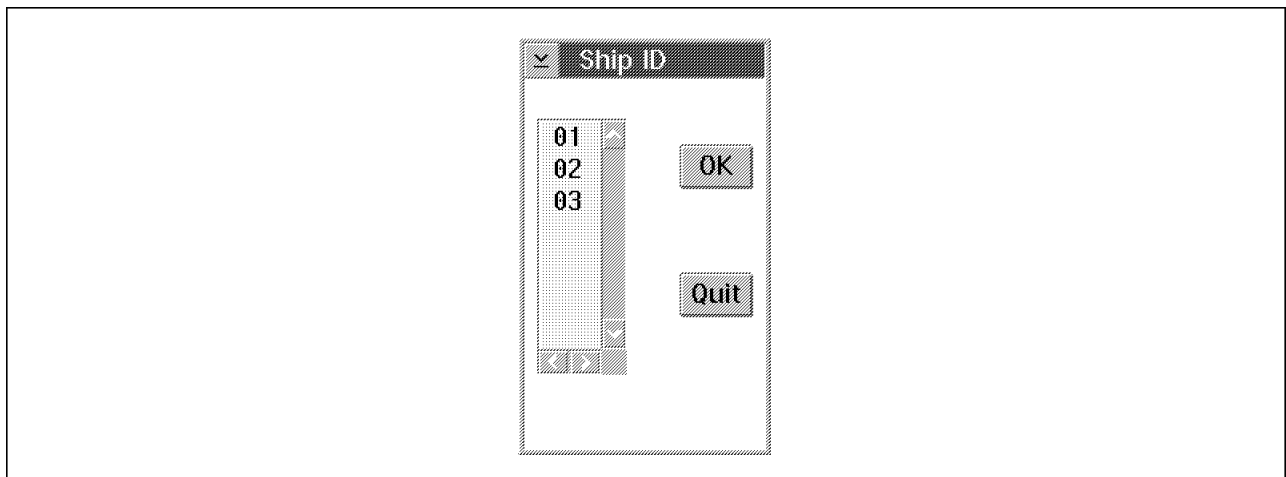


Figure 13. Ship ID Window

You are ready to add the order to the main list box. Click on **OK** and place the order by clicking on **Place Order**.

In summary, there is one main window which has five children at level one, That is, each child has the main window as its parent (Figure 14).

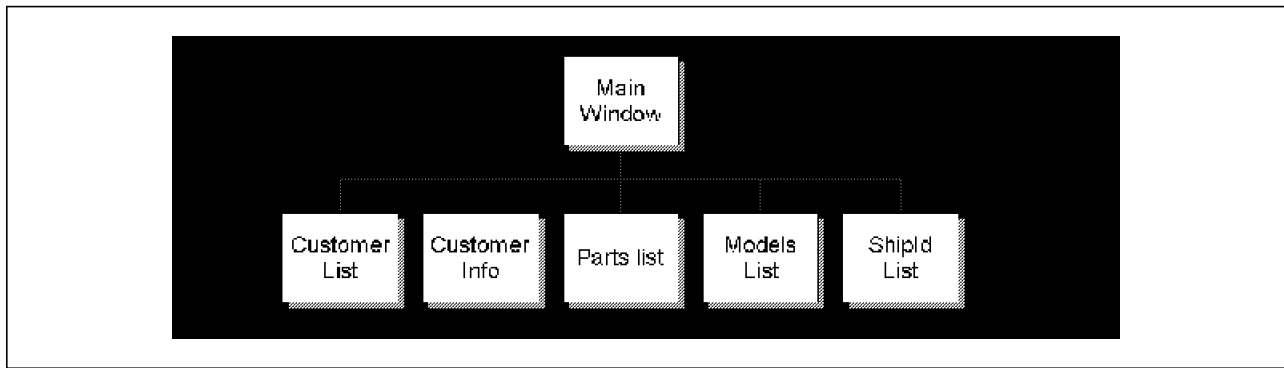


Figure 14. The Celdial Order Entry

Chapter 3. Migration Utility Overview

3.1 Background

The Migration Utility program is designed to help you migrate the GUI section of the MF Dialog System. The objective is to avoid the long and tedious task of recreating, one by one, all of the dialog components that have already been created for the Dialog System.

The utility translates all of the GUI components individually, and where possible, tries to maintain their properties. Certain properties are impossible to convert initially because they are not directly supported, but often these properties can be translated by adding a few statements to the program.

3.2 Utility Description

The Migration Utility was written in REXX using VX-REXX from Watcom. Details about installing the utility and suggestions about customizing it can be found in Chapter 8.

The utility needs an input file (.TXT) from the MF Dialog System. To produce this file, you must go into the Dialog System and, after opening the application to be migrated, you must:

- Select the **File** option from the main DS window.
- Select **Export** from the **File** option.
- From the Dialog System Export window,
 - Select **file** as the destination.
 - Select all of the items in the Sections to Export group box.
 - Select all of the items in the Objects to Export group box.

At this point you have all of the needed input. You need only to invoke the utility by double-clicking on the **MF2VDE** icon. The Micro Focus DS to IBM VDE Migration Utility window appears (Figure 15).

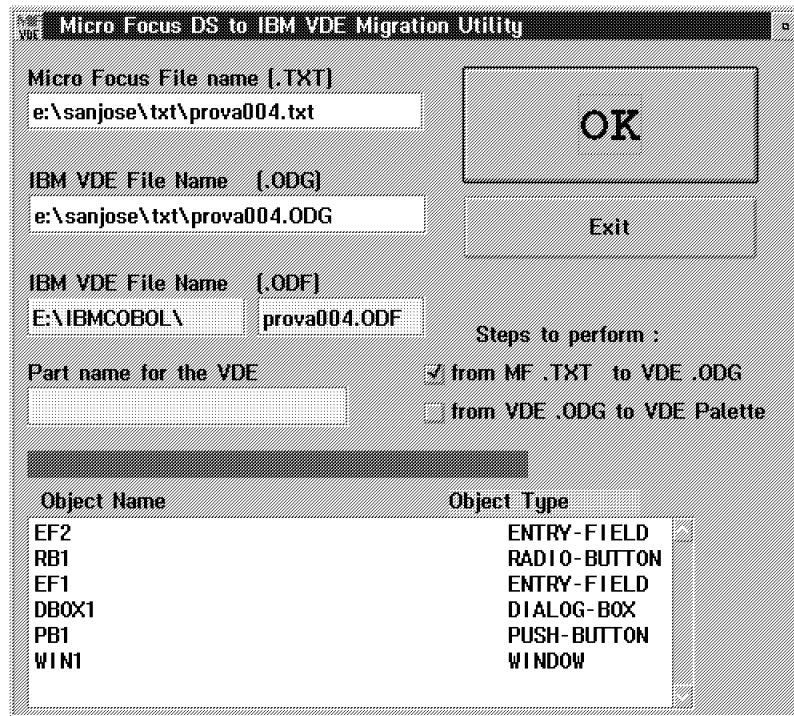


Figure 15. Micro Focus DS to IBM VDE Migration Utility Window

The migration process has two steps. The first step parses the Micro Focus .TXT file (Figure 15) and produces an intermediate file with .ODG. The second step reads the .ODG file and produces an .ODF file, and also adds to the VDE catalog the entire application dialog being migrated, as a single part, with the name you specified in the *Part name for the VDE* entry field.

In the VDE, parts (objects in the Dialog System vocabulary) are stored in files with extension .ODF. Even if each project in the VDE has its own .ODF file, application layout information need not be written. The .ODF file will contain this information after you create the application, drag the iconized part into the application workframe, and save the application.

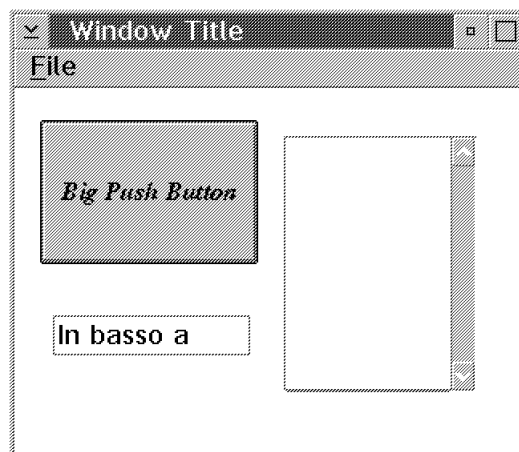


Figure 16. Sample Micro Focus Dialog System Window. This window migrates with the utility.

Here is the .TXT file obtained from MF, related to the window in Figure 16:

```
Screenset Details
  First-Window WIN1
  Decimal-Char '.'
  Comma-Char ','
  Currency-Sign '$'
  Error-File "derror.err"
  Icon-File ".\\ds.icn"
  Style FLAG-88 ANSI EMU-PORTABLE
End Details

Font-Record Stylename {FONT-001}
  Typeface "Helv Italic"
  Pointsize 8
  Attributes BITMAPPED MONOSPACED BOLD ITALIC UNDERLINE
End-Record

Font-Record Stylename {FONT-002}
  Typeface "Times New Roman Bold Italic"
  Pointsize 10
  Attributes VECTOR PROPORTIONAL
End-Record

Font-Record Stylename {FONT-003}
  Typeface "Tms Rmn Italic"
  Pointsize 10
  Attributes VECTOR PROPORTIONAL
End-Record

Object WIN1
  Type WINDOW
  Parent DESKTOP
  Start (440,6)
  Size (1272,733)
  Display "Window Title"
  Style SIZE-BORDER TITLEBAR SYSTEM-MENU MINIMIZE MAXIMIZE CLIPPED
End Object #WIN1

Menu
  Parent WIN1
  Item {NoName}
    Level 0
    Display "File"
  End Item
  Item {NoName}
    Level 1
    Display "Exit"
    Key "F3"
  End Item
End Menu

Object LB1
  Type LIST-BOX
  Parent WIN1
  Start (672,96)
  Size (480,512)
  Style DISABLE-HORIZONTAL
End Object #LB1
```

```

Object PB1
    Type PUSH-BUTTON
    Parent WIN1
    Stylename {FONT-002}
    Start (64,224)
    Size (544,288)
    Display "Big Push Button"
    Style DEFAULT
End Object #PB1

Object EF1
    Type ENTRY-FIELD
    Parent WIN1
    Start (96,512)
    Size (472,64)
    Style AUTOSCROLL BORDER CENTERED UPPER AUTOSKIP
    Picture x(10)
    Display "In basso a destra"
End Object #EF1

Global Dialog
    Event ESC
        RETC ;
    End Event # ESC
    Event CLOSED-WINDOW
        RETC ;
    End Event # CLOSED-WINDOW
End Dialog

```

A portion of the information contained in the above file will be ignored by the Migration Utility, because the program is not currently in a position to translate an equivalent in the VDE. Therefore, you must do some of the work for those statements manually. This is true for statements relating to the event logic: in the above example, the statements between Global Dialog and End Dialog are ignored. A log file is produced containing information regarding what the process has been unable to migrate.

The following is an example of how that file should look:

```

----- OBJECT : WIN1 (FrameWindow)
property CLIPPED cannot be implemented automatically.
----- OBJECT : {NoName} (MenuItem)
property {NoName} cannot be implemented automatically.
----- OBJECT : {NoName} (MenuItem)
property {NoName} cannot be implemented automatically.
----- OBJECT : LB1 (ListBox)
----- OBJECT : PB1 (PushButton)
----- OBJECT : EF1 (EntryField)
property UPPER cannot be implemented automatically.

```

The result of the process (the .ODG file) should have a format similar to the following:

```

B 100 0
W FVDESFrameWindow V 318 V 294 V 110 V 2 V 00000800 V 00000033 D D V WIN1 D D D D D D V Window@Title D D D
W FVDESMenuBar D D D D D V 1 V 1 D D D D D D D D D D D D D D
W FVDESMenuItem D D D D D D D V NoName D D D D D D D V File D D D D D
W FVDESSubmenu D D D D D V 1 V 1 D D D D D D D D D D D D
W FVDESMenuItem D D D D D D D V NoName D D V 34 D D D D V Exit D D D D D
X
X

```

```

X
X
W FVDESCanvas D D D D D V 1 V 1 D V 1 D D D D D D D D D
P FVDESListBox V 120 V 170 V 168 V 42 D D D D V LB1 D D D D D D D D D D D D
P FVDESPushButton V 136 V 96 V 16 V 121 V 00000400 D D D V PB1 D D D D D D V Big@Push@Button V Times@New@Roman@Bold@Italic V 0 V 10 D
P FVDESEntryField V 118 V 21 V 24 V 73 V 00000010 D D D V EF1 D D D D D D V In@Basso@a@destra D D D D D D D D D D D D D D D
X
X
E

```

This is the only format which can be used to import a new part into the VDE as a user-defined part. A minimum of two steps are needed for such an import.

Any new part that is imported with this approach must be a collection of existing parts; that is, you can create a new part as an entry field and a static text and set certain properties to make them more user friendly. You cannot, however, add a new part (for example, Draw a Pie) that is not a composition of existing parts.

3.3 How to Use the Migration Utility

The only input needed is in the .TXT file, but before beginning you must fill in another field name using the extension .ODG. We suggest using the name and path of the .TXT file. To do so, after filling in the .TXT field, move to the next field and the utility will fill in the entry field automatically. Click **OK** to complete Step 1 (Figure 17).

Note: Be sure you do not select both boxes at once; the steps should be performed one at a time.

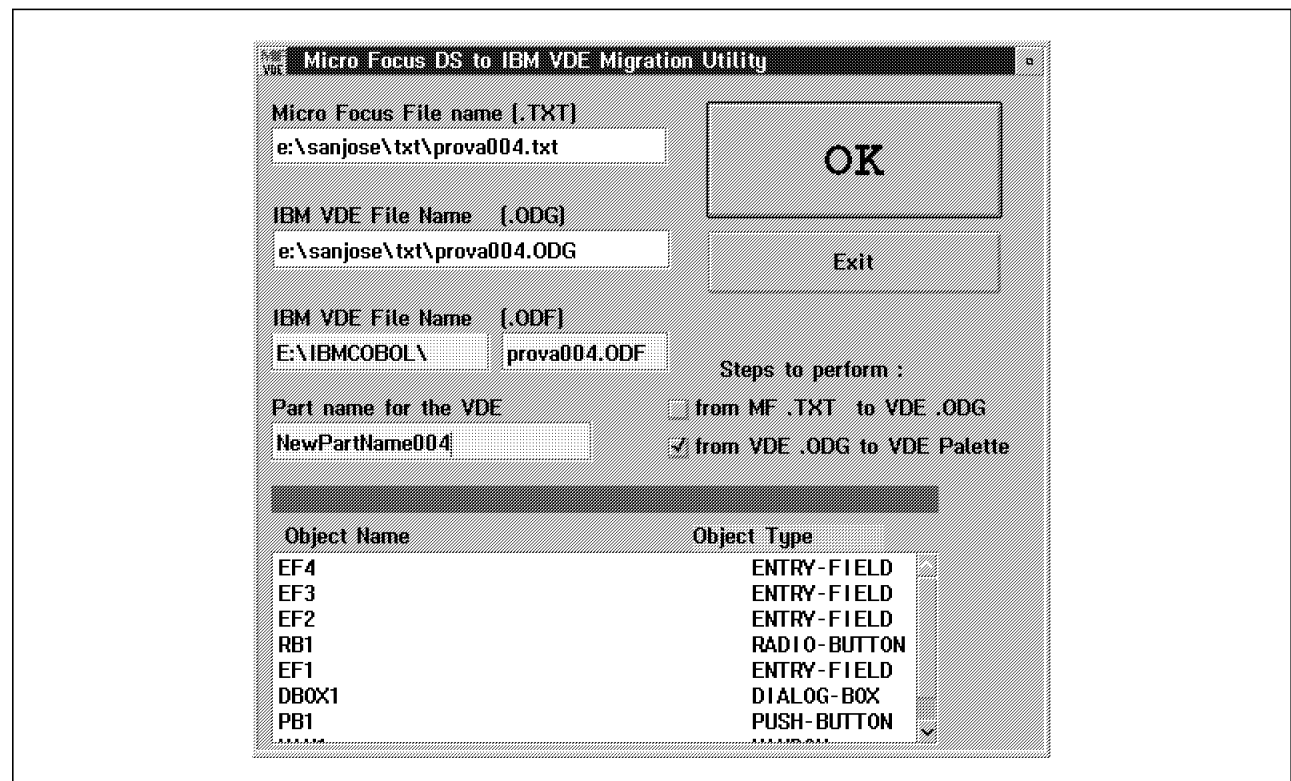


Figure 17. Micro Focus DS to IBM VDE Migration Utility Window

During the process, if the input file has rows longer than 79 characters, the procedure will display a message similar to:

File Restructuring needed. Just continue

and you must click on **OK**. This means that the procedure is taking additional time to rewrite the input file in order to make it more readable.

After each step completes, you receive the message shown in Figure 18.

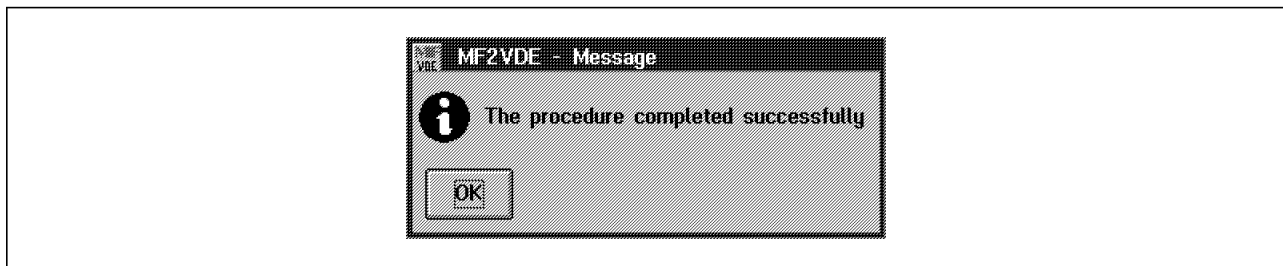


Figure 18. MF2VDE - Message Window Migration Utility

When the first step is completed, you will find two files in the task list, a .LOG and an .ODG file.

The .ODG file is an informational file; you may want to check whether all the objects are included.

The .LOG file is important because it contains a list of the processed objects and the properties for each object which cannot be transferred automatically. You should look at this file carefully, object by object, and note the properties that have been left out. This process must be done after the part is imported into the new object.

To begin Step 2, do the following (Figure 17):

- Ensure that the IBM VDE is not open. You cannot add a new part to the catalog if the VDE is active.
- Deselect the first check box and select the second one.
- Specify the .ODF file name.
- Specify the part name you are adding to the catalog..

A valid part name is 1 to 23 alphanumeric characters (A-Z, 0-9) long and contains at least one alphabetic character. A hyphen is also valid, but it may not be the first or last character. Blank characters are not allowed.

For the .ODF file name, we suggest you use the file name proposed by the utility.

At the end of the process, you should hear a triple beep confirming that the process has completed without errors. Otherwise, you will receive a message window with error codes. Again, the Migration Utility is meant to handle a fair number of MF Dialog System designs but not all DS system designs can be migrated. Differences between DS and VDE will require manual migration.

At this point you are ready to open the VisualAge COBOL project. Open the tools palette and find, in the user-defined page, the part you added to the catalog (Figure 19). Now you need only drag the selected part into the VDE frame, drop it, and wait a few seconds until all of the MF DS windows and objects are shown.

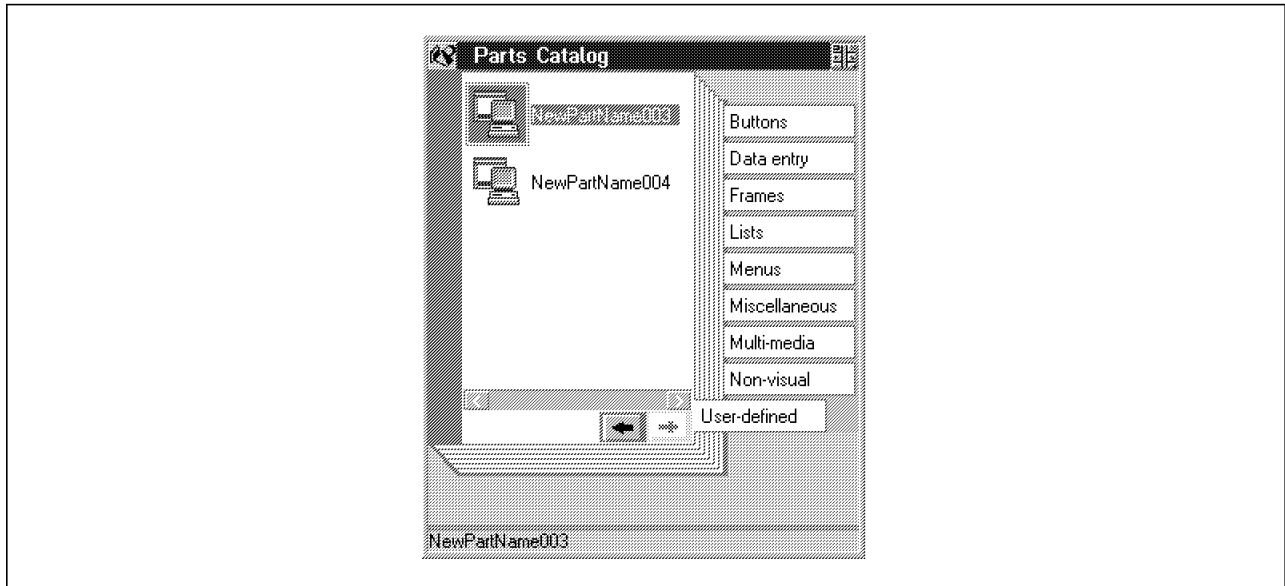


Figure 19. Parts Catalog Window of the Migration Utility. Shows the new user-defined part

Before executing the build command, you should test to ensure that what you have created is correct. Do this by using the steps that clean up the VDE environment:

- From the project you have just opened, delete the window automatically added by VDE. Its name is usually FRA0000B and it is entitled "Window with canvas"
 - Take a look at the .LOG file and look for the string:


```
>>>>>>>> part name too long.
```

 If you find it, the string should continue with the part name. In this case the string is:


```
>>>>>>>> part name too long: OEMAIN-ORDER-DETAIL-PARTPRICE.
```

 If you read the following two rows, you find:


```
>>>>>>>> it must be at least 23 characters, CORRECT IT ! in the VDE.
          ----- OBJECT : PB1-LUNGOPIUDIVENTITREBYTEPER (PushButton)
```

 This message informs you that the object which needs to be changed is a push button. To find the parent of this object you must go back through the .LOG file until you find the first FrameWindow. This is its parent.
 - Go into the VDE and find the push button in the first FrameWindow. Change its part name to a length less than 23 characters.
- Repeat the above cycle to find any other too-long part names.
- When finished, open the source code, save it, close its window, and save the project.
- You are now ready to build.

If the build completes successfully, you can begin changing and adding the properties that the utility has not migrated.

Chapter 4. Comparison of MF Dialog System Objects and VisualAge COBOL Parts

This chapter describes the objects used by the MF Dialog System and compares them with the objects (parts in the IBM environment vocabulary) belonging to the VDE. For each object, we stress only those differences that are not portable from the MF Dialog System to the VDE.

4.1 General Differences

The only aspect common to all the objects but not supported by the Migration Utility is color. The Migration Utility does not support colors as attributes. To set colors, you must change them object by object in the colors page of the property notebook.

4.2 Windows

Fortunately, there are no complex differences between MF Dialog System windows and VDE windows.

Most of the properties in the MF window (Figure 20) can be found in the VDE object window (Figure 21). In the VDE object window you can also specify which behavior your window needs when you hide or minimize it.

The only window property not supported by VDE is *clipped*, which is a parameter that allows you to choose whether or not the secondary window is clipped within the primary window. A window is clipped if the window information is truncated at the border of the parent window.

The following is a list of the properties you must add to the VDE project manually:

- *System-Position*

This property has a corresponding property in VDE, *DEFINED by SYSTEM*, which can be obtained from the window notebook settings in the startup page.

- *Relative-Position*

This property has a corresponding property in VDE, *DEFINED By APPLICATION*, which can be obtained from the window notebook settings in the startup page.

The menu bar object for the VDE window is not in the settings notebook of the window properties, but in the Parts Palette window. The menu bar object is not a window property but a different object.

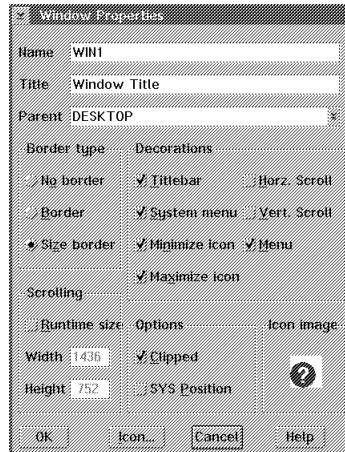


Figure 20. Window Properties Window for MF Dialog System

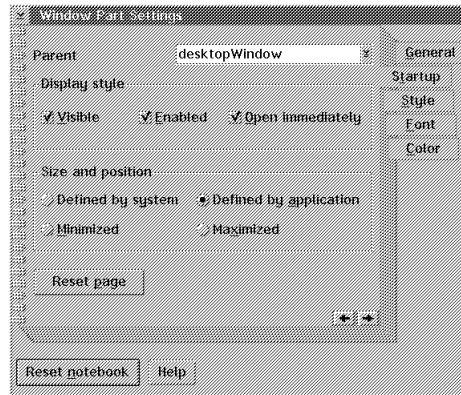


Figure 21. Window Part Settings Window for IBM VDE

4.3 Menu Items

The following is a list of properties not supported by the VDE:

- *Hide-Accelerator*

This property allows hiding the accelerator description from the text of the menu item. For example, if you associate F3 with the exit menu item, turning this flag ON shows the string Exit F3; otherwise, you receive the string Exit. This can be fixed in the VDE environment by changing the menu item label manually.

- *Context-Only*

This property causes the menu item to work in context usage only.

In the MF DS, the shortcut key for the menu item object is a part of the properties of the object (Figure 22). This is also true in the VDE (Figure 23), but here it is not possible to specify it manually.

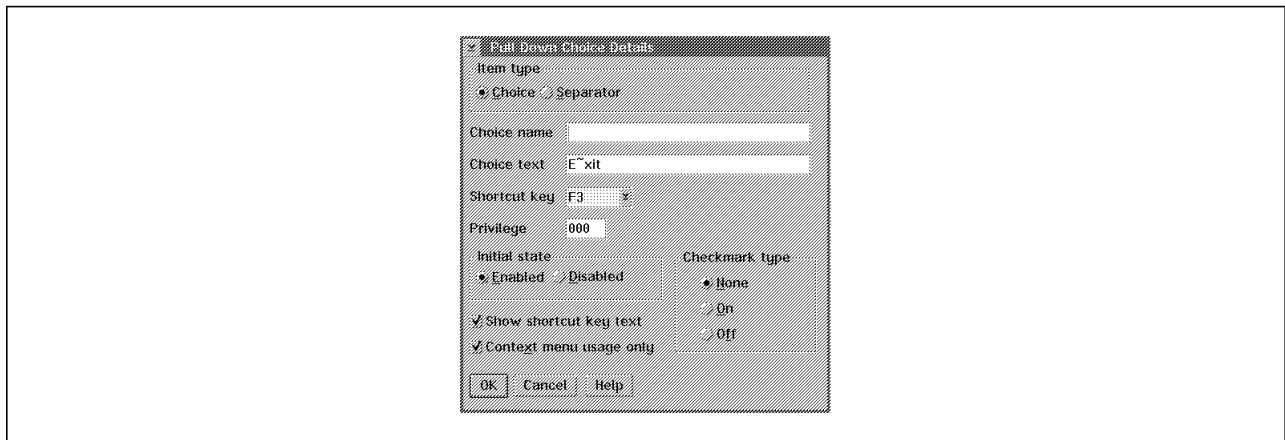


Figure 22. Pull Down Choice Details Window for the MF Dialog System

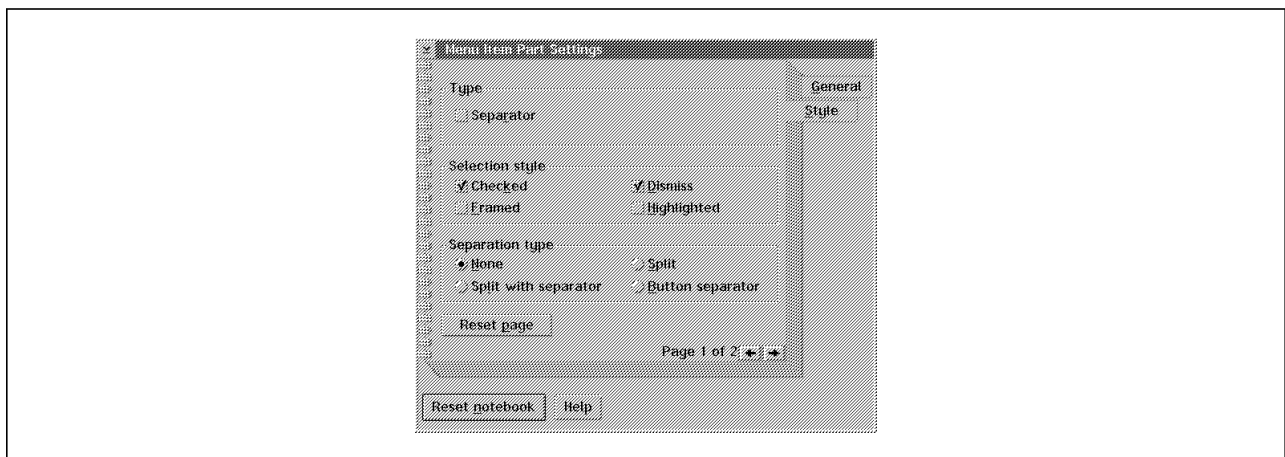


Figure 23. Menu Item Part Settings Window for IBM VDE

4.4 Entry Fields

For the entry field objects, the difference that appears immediately is that for the MF DS you must define a COBOL variable (MasterField) related to the object (Figure 24) This variable represents the bridge between the Dialog System and the COBOL program.

Micro Focus does validations on the entry fields based on the format of the field and on values the user specifies. This kind of validation can only be partially translated into the VDE (Figure 25). The following is a list of the validations that VDE cannot completely support:

- *Null Disallowed*

Nulls are spaces in alphabetic or alphanumeric fields and zeros in numeric fields. If the content of the field contains a null value, this property forces you to change the value. To make this available in an VisualAge COBOL environment, you must add code to the CHANGE or LOSTFOCUS events related to that particular entry field.

- *Date validation*

This checks whether the *date* field contains a valid date. You must add code to the LOSTFOCUS event in order to check whether the date is valid.

- *Range-tables*

This property permits the definition of validation tables. The MF Dialog System allows you to define more than one range or value. The VDE allows you to define one validation range and an alternative to the validation value. If you want to accept values such as zero or those between 100 and 200, you cannot use the validation property only, you must add code to the LOSTFOCUS event. For the MF Dialog System you can define more than one range-table or value in the OR field.

- *BLANK-WHEN-ZEROS*

This validation is for numeric items only. It inserts spaces rather than zeros in the entry field. For an equivalent property in the VDE, code must be written.

The following properties are not supported by IBM VDE:

- *REQUIRED*

This forces you to partially fill the entry field; it will not allow the entry field to be empty.

- *FULL*

This forces you to fill the entry field completely.

- *UPPER*

This automatically translates the content of the entry field into upper case.

- *LOWER*

This automatically translates the content of the entry field into lower case.

- *AUTOSWIPE*

This selects all of the data associated with a field you tab to that field. Any data you type in replaces the selected data.

- *FIT-PICTURE*

This property allows the Dialog System to automatically define an entry field length based on the length of the string defined in the MasterField parameter. Because VisualAge COBOL does not use any predefined variable related to the entry field, this property cannot be migrated.

The following are MF properties and their corresponding names in the VDE:

- Display field --> Visible
- No echo --> Masked
- AutoSkip --> AutoTab
- Border --> Margins

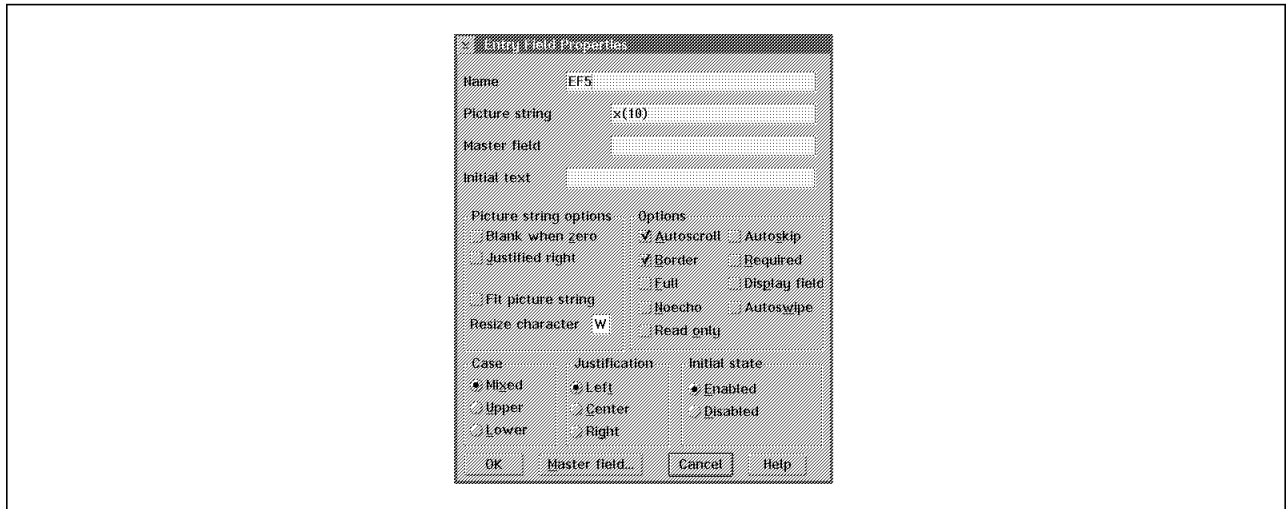


Figure 24. Entry Field Properties Window for MF Dialog System

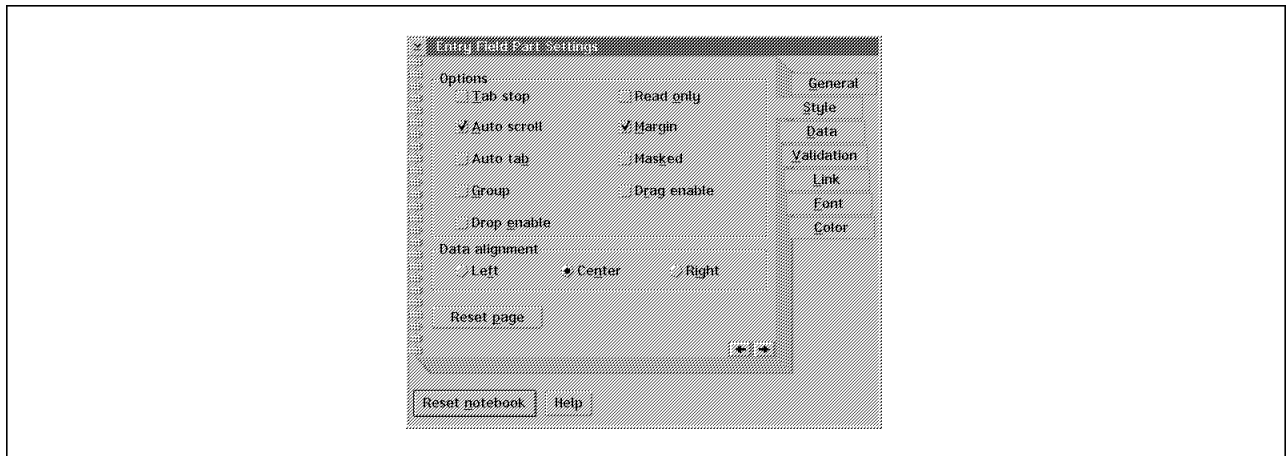


Figure 25. Entry Field Part Settings Window for IBM VDE

4.5 Push Buttons

All of the properties of the MF DS push buttons (Figure 26) are a subset of the IBM VDE push button properties. Therefore, they do not pose a migration problem (Figure 27).

One MF Dialog System property which cannot be automatically migrated and is related to the dimension of the push button object is *fit-text*. With *fit-text*, the push button does not need any dimensioning; it remains slightly larger than the related text. This property can be fixed manually.

Notes:

- In the IBM VDE, you can associate a shortcut key with a push button.
- In the MF Dialog System, one push button object has the ability to be a graphic or text push button, while the VDE has two different objects, a push button and a graphic push button.

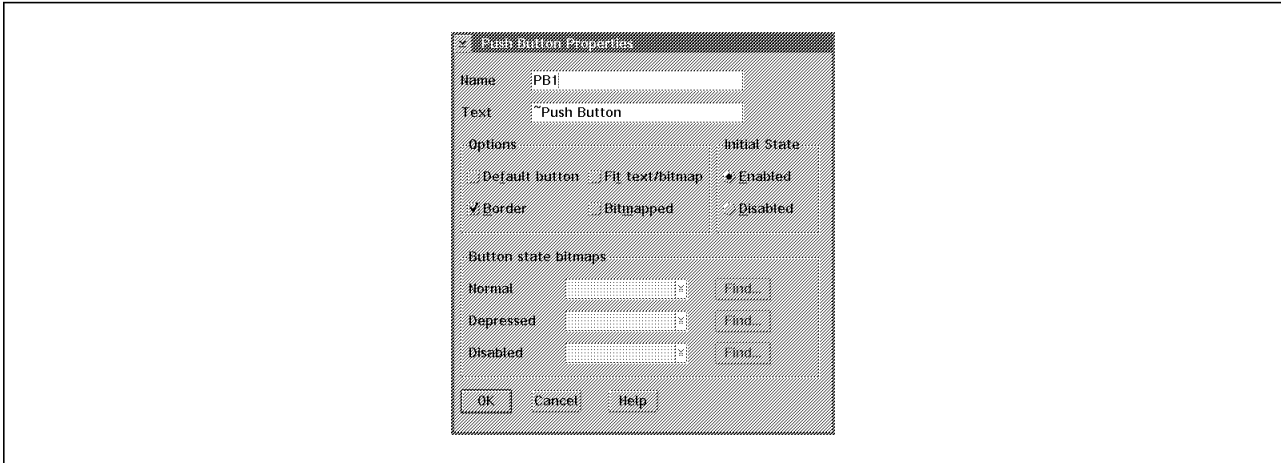


Figure 26. Push Button Properties Window for MF Dialog System

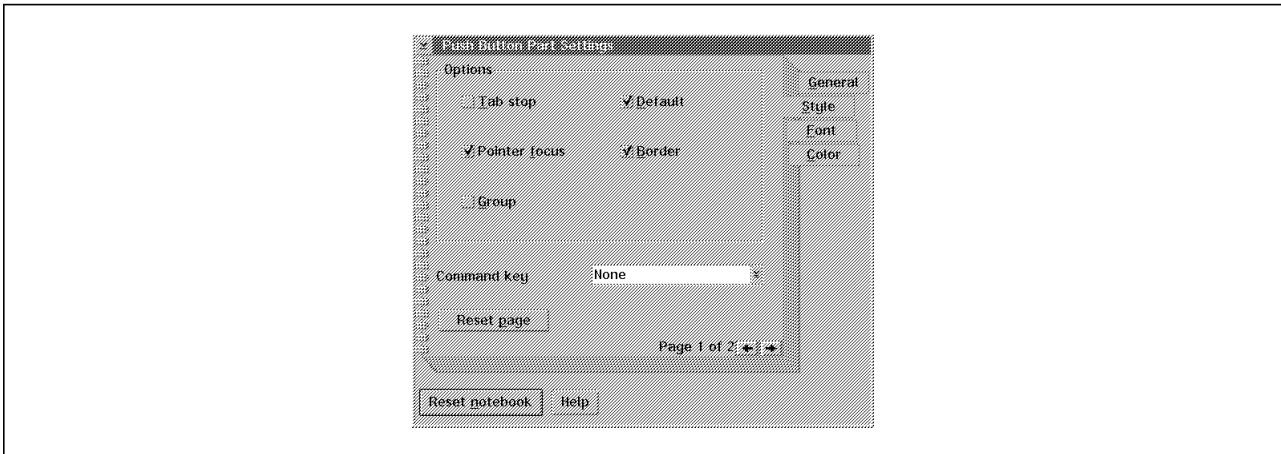


Figure 27. Push Button Part Settings Window for IBM VDE

4.6 List Boxes

For its list box entry field, the MF Dialog System requires the use of a Masterfield with the related consequences (Figure 28).

The VDE (Figure 29) does not automatically support the *blank-when-zeros* property.

There are no major list box differences. Use of the *associated with group* property in the MF DS is easy to translate into the IBM VDE syntax by using the same COBOL structure when you want to populate or, in general, act on it.

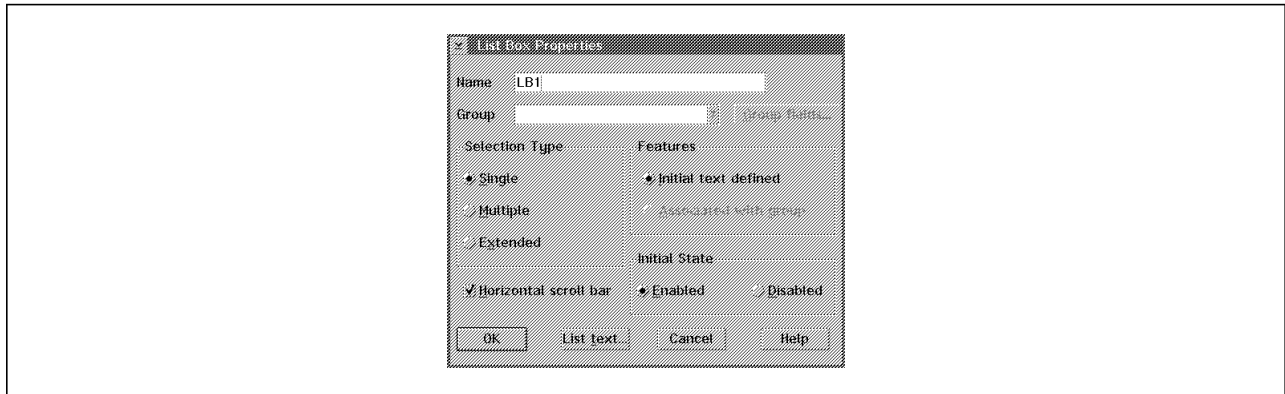


Figure 28. List Box Properties Window for MF Dialog System

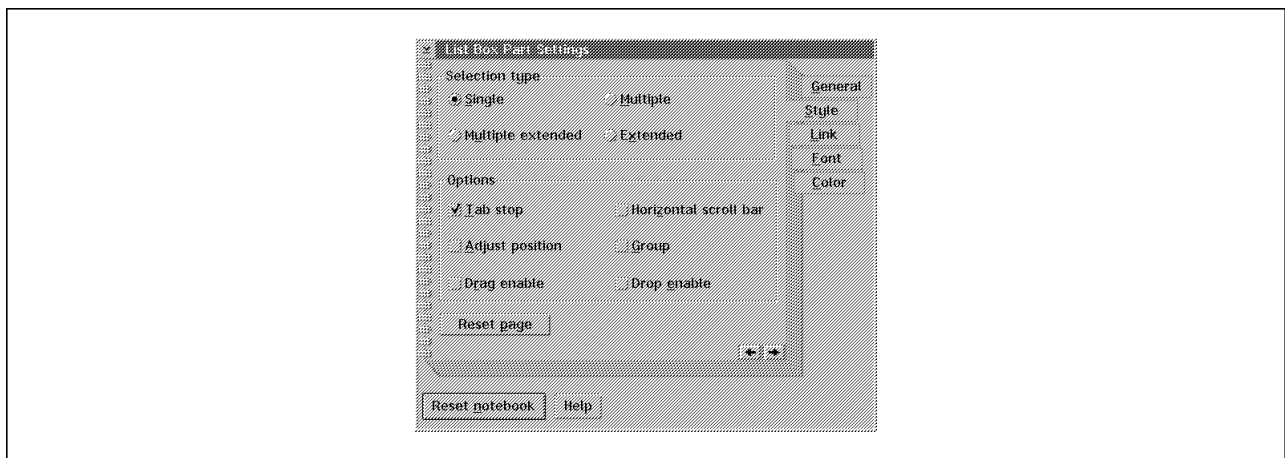


Figure 29. List Box Part Settings Window for IBM VDE

4.7 Multiline Entry Fields

Multiline entry fields pose no migration problems. The MF DS multiline entry fields are a subset of the IBM VDE entry field properties.

4.8 Radio Button

The radio button poses no migration problems. The MF DS radio button properties are a subset of the IBM VDE radio button properties.

4.9 Selection or Combination Boxes

The MF DS selection box corresponds to the IBM VDE combination box.

In the MF DS environment, you must specify whether selection box items must be in upper case, mixed case, or lower case. This is not needed in the IBM VDE. You must also define which type of value you are using in the selection box (for example, x(10) or 9(4)) and you may specify that items should be “blank when zero” or “justified right.”

The remaining properties are similar.

4.10 Static Text

The static text object does not have any properties in the MF DS environment and thus poses no problems for the migration.

4.11 Group Boxes

The MF DS group box and the IBM VDE group box have the same properties; therefore, they are equivalent.

4.12 Bitmaps

The MF DS bitmap object contains a subset of the IBM VDE bitmap object properties.

The bitmap object in the MF DS export file contains no information about the size of the bitmap it displays because, unlike IBM VDE, the MF DS does not allow scaling of pictures. As a result, after migration the utility places the bitmap in the correct starting position, as defined in MF DS. However, its size could cover another object in the window. To remedy this, use manual scaling.

4.13 Notebooks

The migration utility does not yet support notebooks.

You must migrate notebooks manually.

4.14 Dialog Boxes

Dialog boxes in the MF DS are considered objects and are not treated as windows. In IBM VDE, dialog boxes are secondary windows and are treated as such. The migration utility recognizes the dialog box and translates it into a secondary window.

4.15 Message Boxes

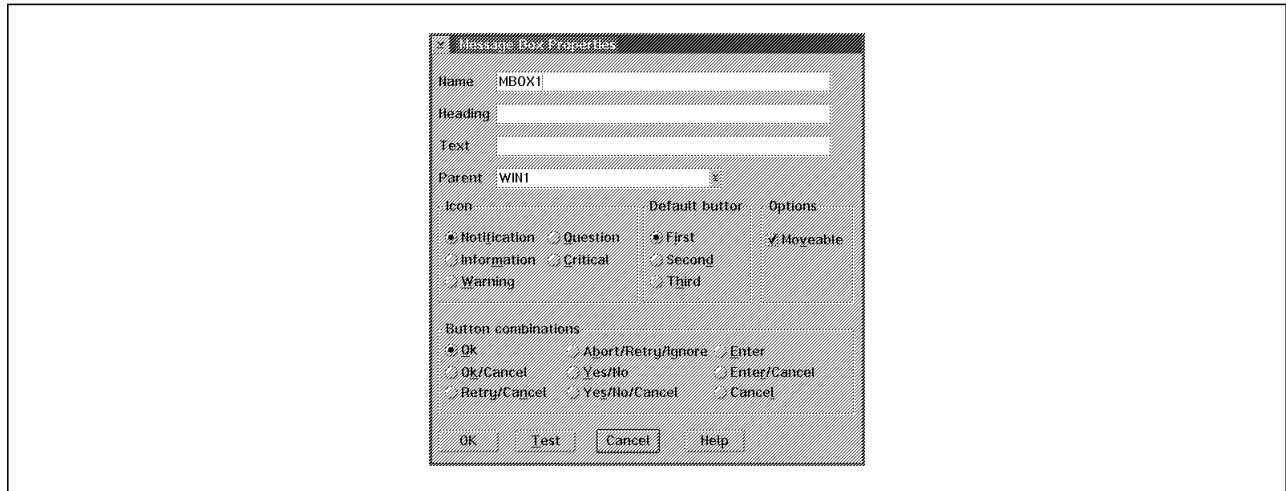


Figure 30. Message Box Properties Window for MF Dialog System

MF DS message boxes (Figure 30) cannot be migrated automatically, but it is possible to define a message box manually. In order to migrate a dialog box use the following method:

- In the log file produced by the migration utility, find the message box you wish to migrate, if any. You should find information similar to:

```
----- OBJECT : MBOX1 (MessageBox)
+++++++ the above object: MBOX1 must be migrated manually
property MOVABLE cannot be implemented automatically.
property BUTTONS(OK) cannot be implemented automatically.
property DEFAULT-BUTTON(1) cannot be implemented automatically.
property NOTIFICATION cannot be implemented automatically.
```

- Next, use other information from the exported MF DS .TXT file. If you look for the string MBOX1, you should receive this type of data:

```
Object MBOX1
  Type MESSAGE-BOX
  Parent OEMAIN
  Style MOVABLE BUTTONS(OK) DEFAULT-BUTTON(1) NOTIFICATION
  Display "Invalid price"
  Message "This part and model cannot to be discounted"
End Object #NO-DISCOUNT
```

- Go to the VDE and, from the action bar, select:
 - Project
 - Messages
 - Create

You should then see the COBOL GUI Designer - Define Message window (Figure 31).

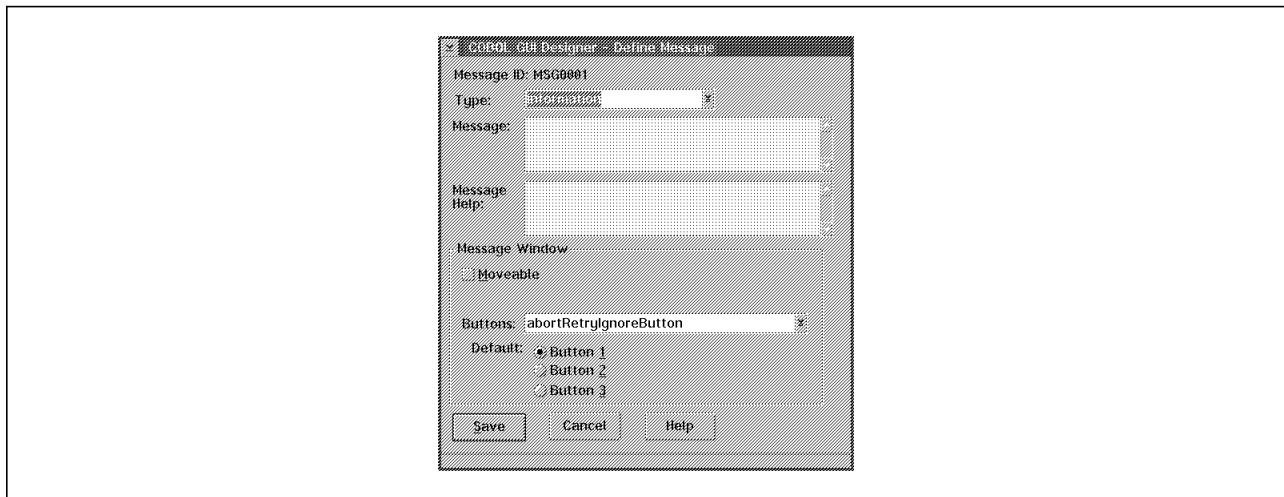


Figure 31. COBOL GUI Designer - Define Message Window for IBM VDE

At this point it is very easy to port the properties into this window.

Chapter 5. Migrating the Sample Application

5.1 A Practical Example

This chapter describes how to migrate the GUI of the Celdial application from the MF Dialog System to the VisualAge COBOL environment. The celdial.txt file is available after you install the Migration Utility as outlined in Chapter 8.

Using the Migration Utility tool (Figure 32), create the following:

- File Celdial.ODG
- File Celdial.ODF
- File Celdial.LOG
- Part name Celdial part for the IBM VDE catalog

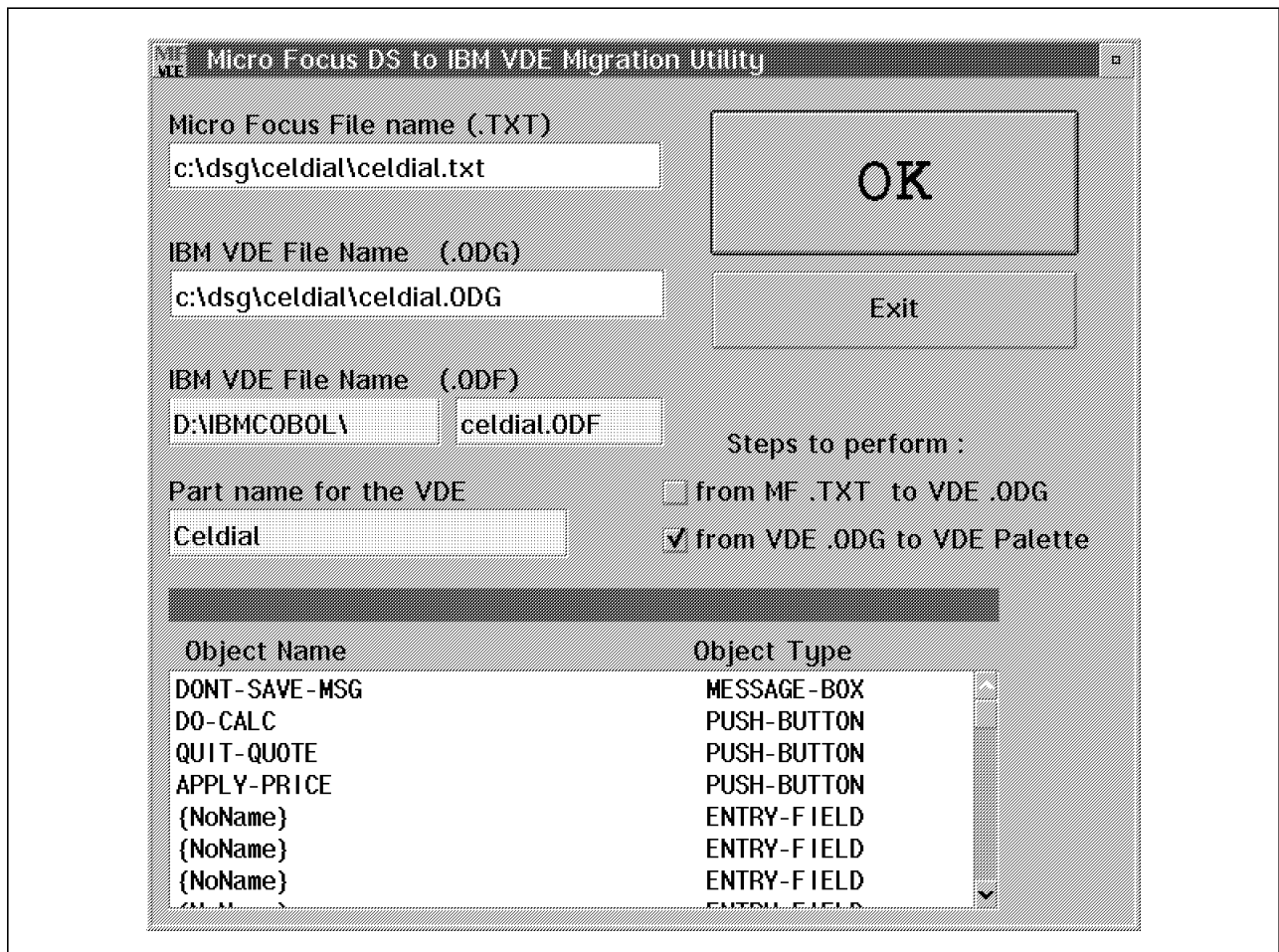


Figure 32. Migration Utility Window. Shows values used for the Celdial application.

After opening the VDE,

- Delete the FRA0000B window from the untitled project.
- Go to the parts catalog notebook and from the user-defined page, select the Celdial part.

- Drag and drop the Celedial part to the project workframe and wait a few seconds until the windows of the Celdial application finish building. A sample of one is shown in Figure 33.

Figure 33. Migrated Ship - ID Window Showing Wrong-Sized Push Buttons.

- In the settings of all the windows, except for the OEMAIN, flag the attribute OPEN IMMEDIATELY to off.
- Look in the CELDIAL.LOG file for part names longer than 23 characters. In this case, we found the following:

As a remedy, we went back to VDE and changed OEMAIN-ORDER-DETAIL-PAR, the name that VDE truncated, into OEMAIN-ORDER-DETAIL-P. This change is needed even if the name has already been truncated.

At this point, start reading the CELDIAL.LOG file row by row, looking for properties you want to keep but that the Migration Utility did not keep.

```
----- OBJECT : OEMAIN (FrameWindow)
property CLIPPED cannot be implemented automatically.
```

The second message you find is:

If you do not want to write code just to keep this property, ignore the message.

The next message in our example is:

```
----- OBJECT : OEMAIN-CUSTNO      (EntryField)
property Null Disallowed            cannot be implemented automatically.
```

In this case, open the part settings notebook and look first at the datatype. A character data type with a length equal to seven, can be fixed in the validation page by specifying Comparison as the validation criterion, with value Not Equal, then typing seven blanks.

The next message is:

```
----- OBJECT : {NoName}           (PushButton)
property FIT-TEXT                   cannot be implemented automatically.
```

This means you need to do some MAKE UP for the dimensions of this push button.

Then, after some properties already treated, you reach these messages:

```
----- OBJECT : NO-DISCOUNT        (MessageBox)
+++++++ the above object: NO-DISCOUNT must be migrated manually
property MOVABLE                    cannot be implemented automatically.
property BUTTONS(OK)                cannot be implemented automatically.
property DEFAULT-BUTTON(1)          cannot be implemented automatically.
property NOTIFICATION               cannot be implemented automatically.
```

First, you must understand the content of the message. Looking in the CELDIAL.TXT file for the NO-DISCOUNT string, you find:

```
Object NO-DISCOUNT
  Type MESSAGE-BOX
  Parent OEMAIN
  Style MOVABLE BUTTONS(OK) DEFAULT-BUTTON(1) NOTIFICATION
  Display "Invalid price"
  Message "This part and model cannot be discounted"
End Object #NO-DISCOUNT
```

You then go to the messages window (Figure 34) and translate those properties as shown.

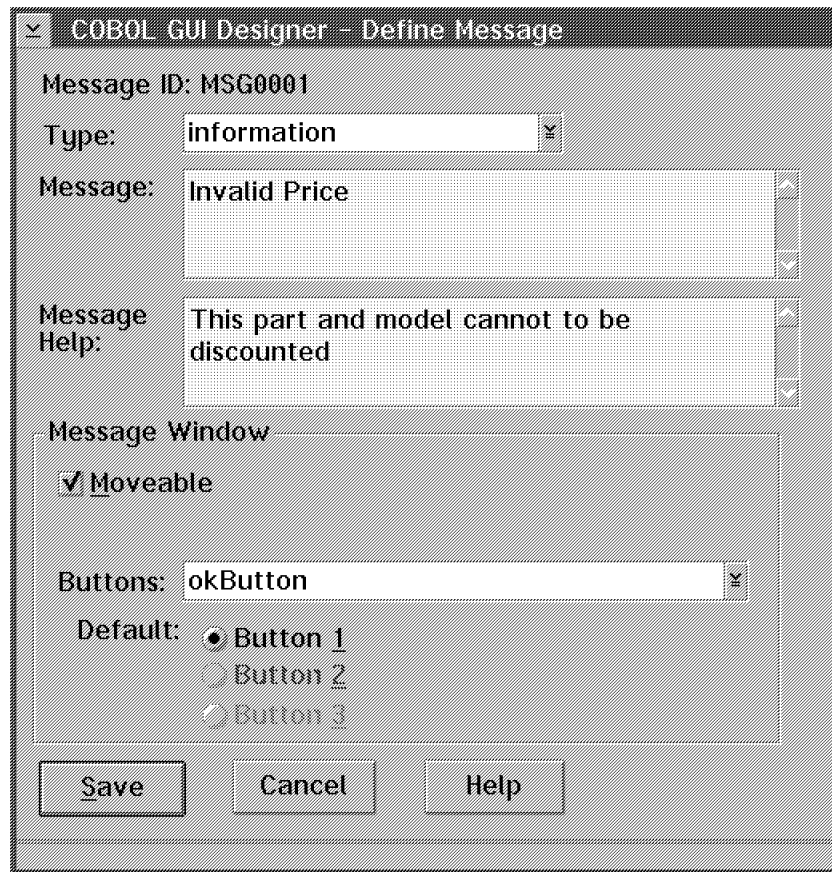


Figure 34. VDE Define Message Window Showing Migration Fixes.

You then reach the end of the CELDIAL.LOG file without finding any other property that needs to be described. Save the project and try a new build operation.

Chapter 6. Embedding COBOL Logic in the Sample Application

Before describing how to embed the COBOL logic, we show how to set the environment to be able to build.

There are several defaults to be changed in the compile-options notebook:

- On the syntactical page (Figure 35) flag the second check box ON, enabling "process BASIS and REPLACE statements."

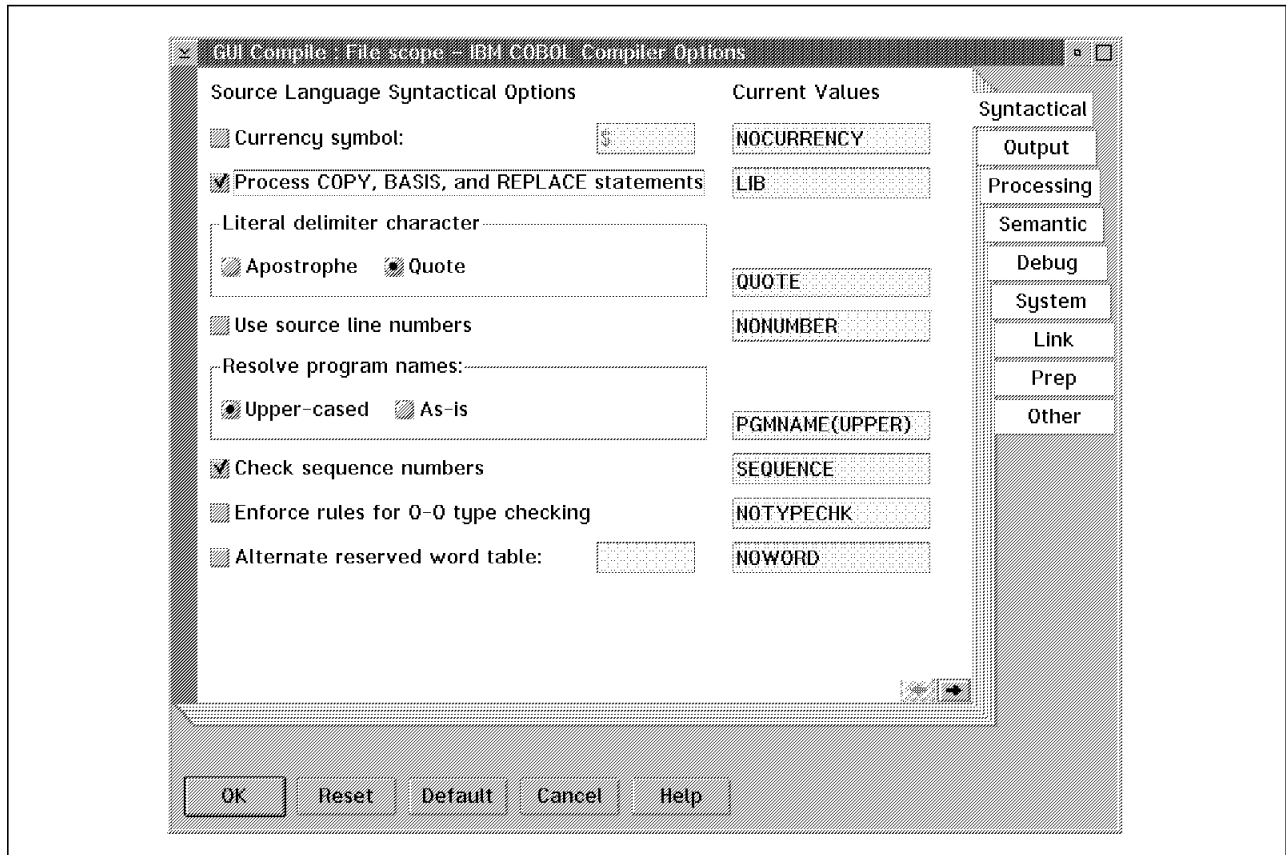


Figure 35. IBM COBOL Compiler Options Window. Shows changes to syntactical defaults.

- In the Debug page (Figure 36), set the check box before the last to ON. This enables the routine "Generate debugging information."

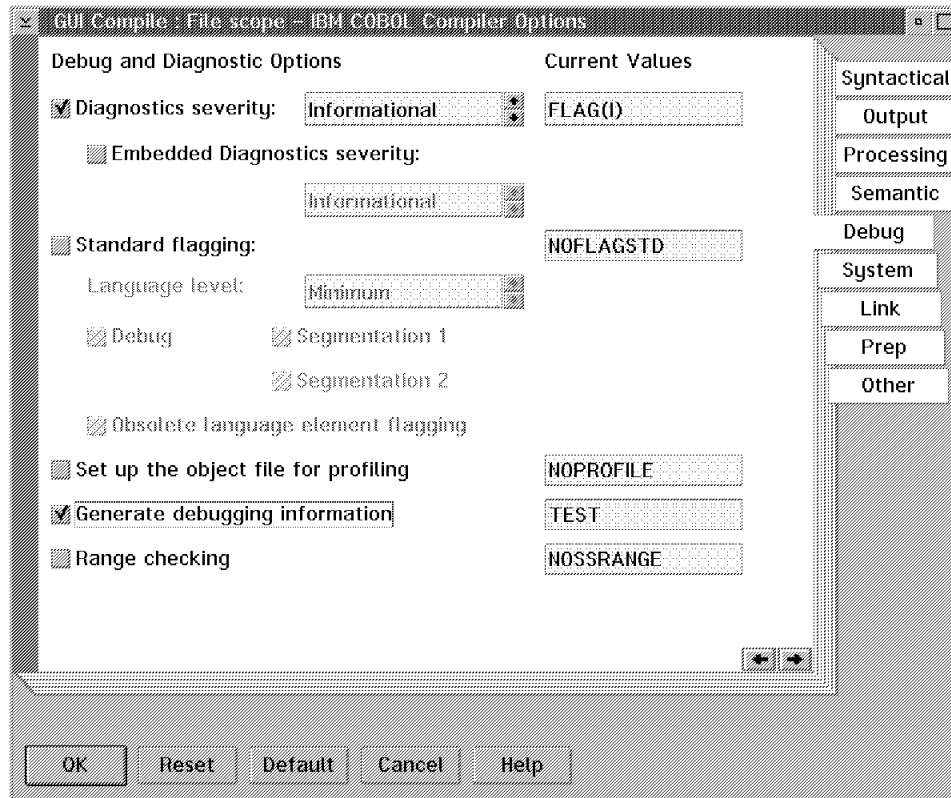


Figure 36. IBM COBOL Compiler Options Window. Shows changes to debugging defaults.

- In the Link page (Figure 37), specify the string:
`d:\sql1lib\lib\db2api.lib`
 In the *library name* entry field.

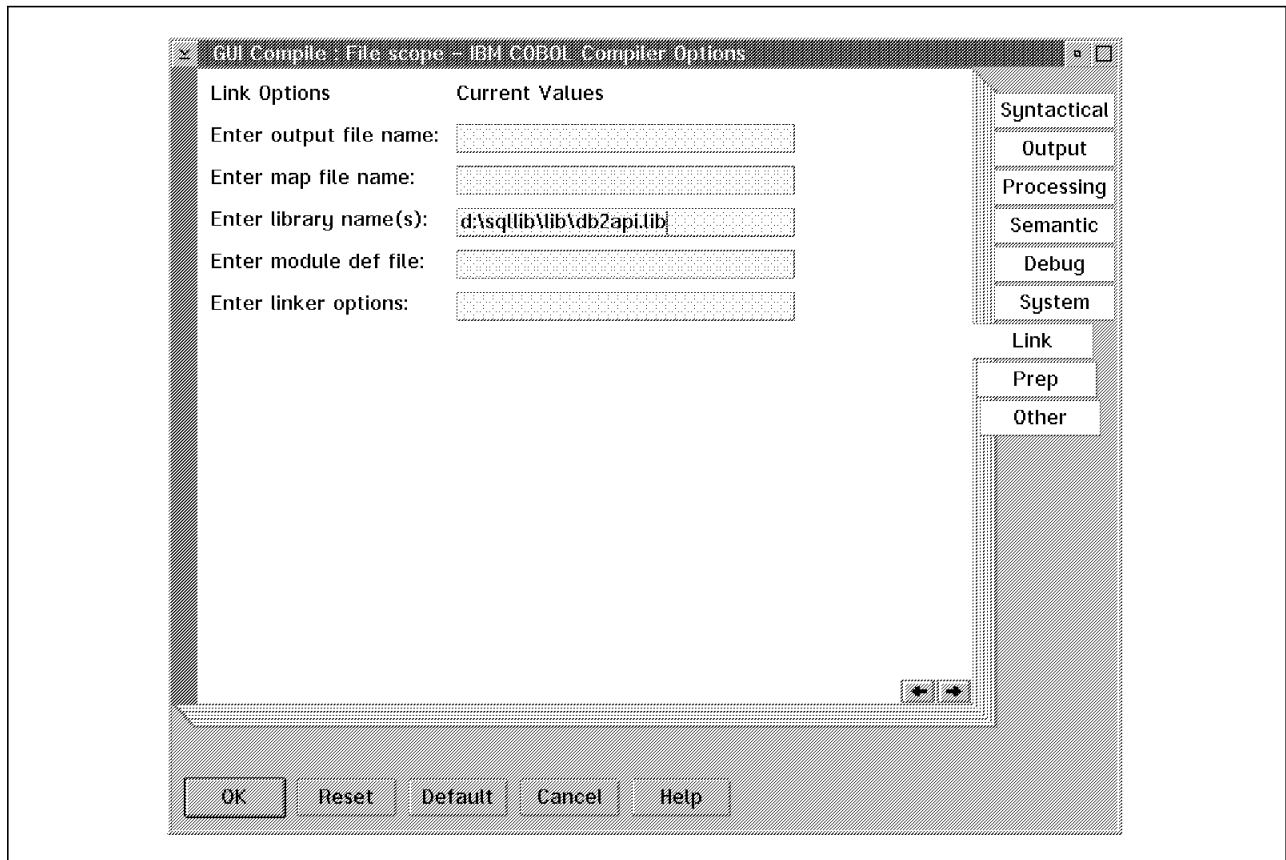


Figure 37. IBM COBOL Compiler Options Window. Shows changes to the link defaults.

- In the Prep page (not shown) specify the database name.
- In the Other page (Figure 38), add a period to the first entry field and flag the first check box ON, as shown.

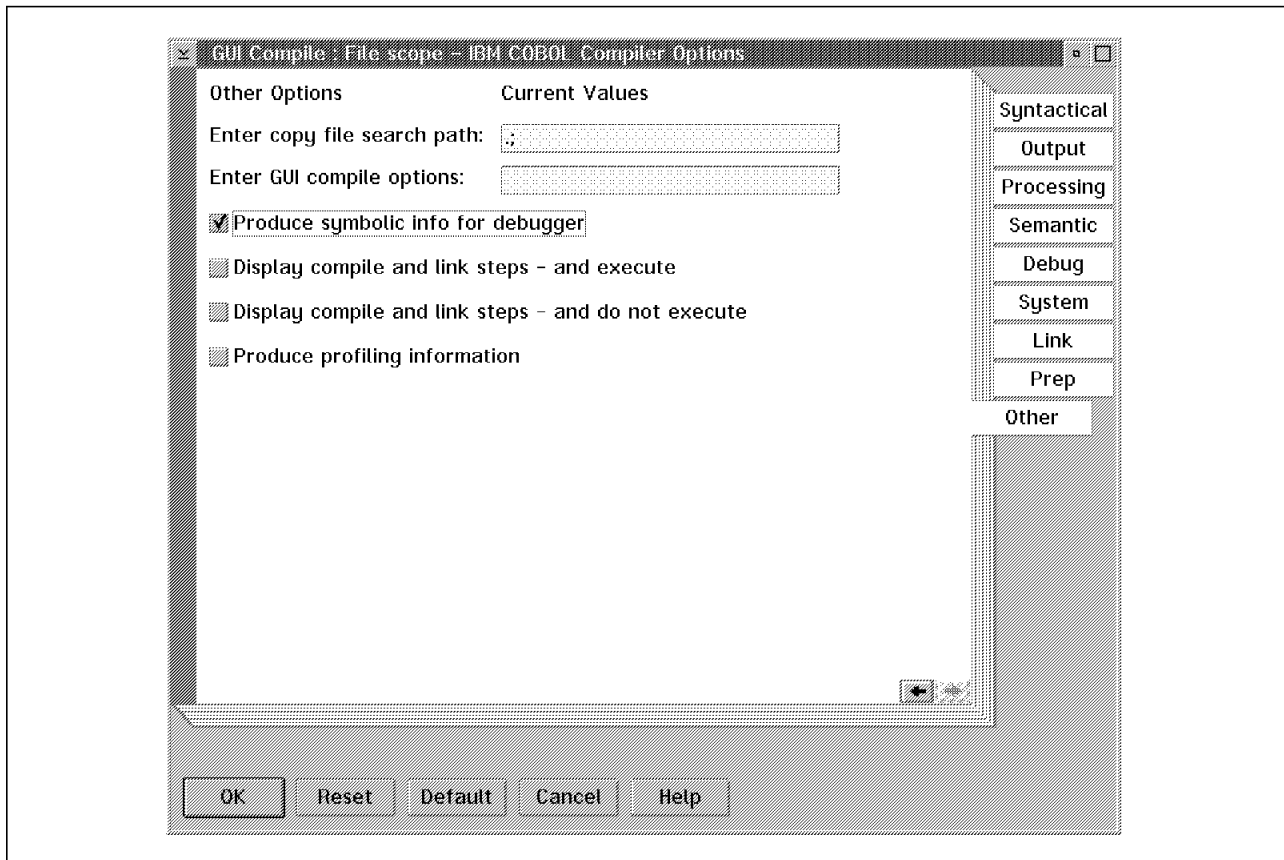


Figure 38. IBM COBOL Compiler Options Window. Shows changes to other defaults.

You are now ready to add the COBOL logic and then compile the whole application.

6.1 Embedding the Old Code (the OLDMAIN procedure)

The first step before adding event logic is to embed the COBOL program into the .CBL file of your project.

To do that, for our example, we defined a procedure called OLDMAIN and decided to embed the whole procedure division in the .CBL file.

We embedded the old COBOL program as follows:

- We copied the whole program at the end of the project source file into the VDE CELDIAL.CBL file.
- We copied the CELDIAL.CPB and SQLENV.CBL copybook files into the project directory.
- We changed **COPY "sqlenv".** to **COPY sqlenv.**
- We deleted the statements immediately preceding the copybooks.
- We deleted the DS-CNTRL.MF copy.
- We moved the two remaining copies to the working-storage section.
- We created a CELDIAL.001 file to store the variable declared by the old program and we added this copybook to the other two.
- We moved the SQL DECLARE statements to the working storage section.

- We discarded the following statements except those flagged with &,

```

    procedure division.
    main-process section.
        perform program-initialize.
        perform program-body until exit-pgm=true.
        perform program-terminate.
*-----*
*
*-----*
    program-initialize section.
* Set handler for signal
    CALL "__SQLGISIG" USING SQLCA.
&    * Connect to db
&    EXEC SQL
&        CONNECT TO celdial in SHARE MODE
&    END-EXEC.
&    if sqlcode not = 0
&        display 'connect error: sqlcode=', sqlcode
&        go to program-terminate.
*    EXEC SQL WHENEVER SQLERROR GO TO EXT END-EXEC.
*    EXEC SQL WHENEVER NOT FOUND GO TO CLS END-EXEC.
    initialize data-block
    initialize pgm-flags
&    move 0 to inx1
&    move 0 to inx2
&    move 0 to inx3
    perform Start-application.
    initialize ds-control-block
    move data-block-version-no to ds-data-block-version-no
    move version-no to ds-version-no
    perform set-date.
    perform load-screenset.
*-----*
* Process the returned user action (in the flags); clear those
* flags and call Dialog System again.
*-----*

```

The statements we kept we then moved between the two statements:

```

    PROCEDURE DIVISION USING CommandLine-Data.
----->
        GOBACK

```

so we could be sure they are executed as soon as the program is run.

- We renamed the program-body section OLDMAIN.
- We then deleted the statements:

```

    perform call-dialog-system

```

and

```

    load-screenset section.

```

- * Specify the screenset to be used and call Dialog System.

```

    move ds-new-set to ds-control
    move "c:\dsg\celdial\celdial.gs" to ds-set-name
    perform call-dialog-system.

```

call-dialog-system section.

call dialog-system using ds-control-block,
data-block.

if not ds-no-error
move ds-error-code to display-error-no
display "DS ERROR NO: " display-error-no
perform program-terminate
end-if.

- When we tried to compile, we encountered these error messages:

2745 IGYPS0008-E A quote or an apostrophe was used as
a character string delimiter. It was not the delimiter option in effect.
The use was accepted.

2756 IGYPS2023-I Paragraph(s) prior to section "OLDMAIN"
were not contained in a section.

2806 IGYPS2008-E A period was required
before procedure-name "PROGRAM-TERMINATE". A period was assumed before
"PROGRAM-TERMINATE".

2807 IGYPA3200-S "STOP LITERAL" clause was
specified under the "THREAD" compiler option.
The statement was discarded.

4161 IGYPS0019-W No COBOL statement was found between periods.

4216 IGYPS2123-S "COMMAND-LINE" was not defined as a mnemonic-name.
The statement was discarded.
Same message on line: 4223

4217 IGYPS2121-S "RESULT" was not defined as a data-name.
The statement was discarded.
Same message on line: 4224

4217 IGYPS2121-S "FUNC" was not defined as a data-name.
The statement was discarded.
Same message on line: 4224

4217 IGYPS2121-S "COMMAND-LIN" was not defined as a data-name.
The statement was discarded.
Same message on line: 4224

4244 IGYPS2015-I The paragraph or section prior to
paragraph or section "CLS" did not contain any statements.
Messages Total Informational Warning Error Severe Terminating
Printed: 14 2 1 2 9

In order to remove these errors we:

- Changed the apostrophe into a double quote at line 2745.
- Added a period to the END-EVALUATE statement.
- Removed the STOP RUN statement.
- Commented out the following statements:

display command-lin-string upon command-line.
call x"91" using result, func, command-lin.

which are supposed to run an external REXX procedure.

- We then tried build again, and got a zero return code from the build,
meaning that we were ready to add events to our project.

Chapter 7. Event Logic Migration to VisualAge COBOL for Sample Application

7.1 Event to Event Logic

To migrate the event logic from one environment to the other, you must translate from the language MF Dialog System uses for the logic in its environment to COBOL. The need for translation is the main reason you must do this migration step completely by hand.

Start from the .TXT file. and look for the global dialog string to start migrating the event that sets the environment. You then follow the order of items found in the .TXT file.

What you find, looking for the global dialog string, is:

```
Global Dialog
  Event ESC
    RETC ;
  End Event # ESC
  Event CLOSED-WINDOW
    MOVE 1 EXIT-PGM(1) ;
    RETC ;
  End Event # CLOSED-WINDOW
  Event SCREENSET-INITIALIZED
    SET-DATA-GROUP-SIZE SAVORDER 0 ;
    MOVE "by Dept W51 of STL" AUTHOR-LINE ;
    SET-COLOR BYLINE "LIGHT RED" "WHITE" ;
    REFRESH-OBJECT OEMAIN ;
  End Event # SCREENSET-INITIALIZED
End Dialog
```

You need not do anything about either the ESC event or the CLOSED-WINDOW event. Both are automatically included in the .CBL file created by VDE. For example, if you look at the content of the "OEMAIN_OEMAIN_DESTROY" event, you find:

```
ENTRY "OEMAIN_OEMAIN_DESTROY"
  USING VDE-CELDIAL.
  MOVE VDE-TERMINATE-APPLICATION TO ACTION-RC.
  GOBACK.
```

where the MOVE VDE-TERMINATE-APPLICATION TO ACTION-RC. statement has the same meaning as MOVE 1 EXIT-PGM(1).

The only thing you then have to migrate is the SCREENSET-INITIALIZED event, which corresponds to the CREATE event of the main window in VDE. In our case, that means the "OEMAIN_OEMAIN_CREATE" event.

The SET-DATA-GROUP-SIZE command is used to set the internal size of a data group in the Dialog System. Since data groups have a related structure in the .CPB file that will be included in the COBOL program, it is easy to understand the translated command:

```
MOVE 0 TO SAVORDER-SIZE
```

The statement MOVE "by Dept W51 of STL" AUTHOR-LINE needs no comment for the translation into COBOL, but you must add some statements to have the same result. Each entry field in the MF Dialog System has a related masterfield, which represents the link between the dialog and the logic, thus between the Dialog System and COBOL, so that modifying the value in the masterfield produces the change in the related entry field, after a refresh object command. In the VDE, to change the content of an entry field, you need to add the following code, but do not need to refresh the object of the whole window:

```
MOVE LENGTH OF AUTHOR-LINE TO Contents-Length
MOVE AUTHOR-LINE TO Contents-String
CALL "setContents" USING BYLINE-HANDLE,
                        Contents,
                        VDE-RC
```

The next statement is the colors setting for the BYLINE entry field. To convert that setting, use the VDE GUI Assistant. In the GUI assistant window, select the OEMAIN window, the BYLINE entry field, and the setBackgroundColor action, then press the **Move** push button displayed in the window (Figure 39).

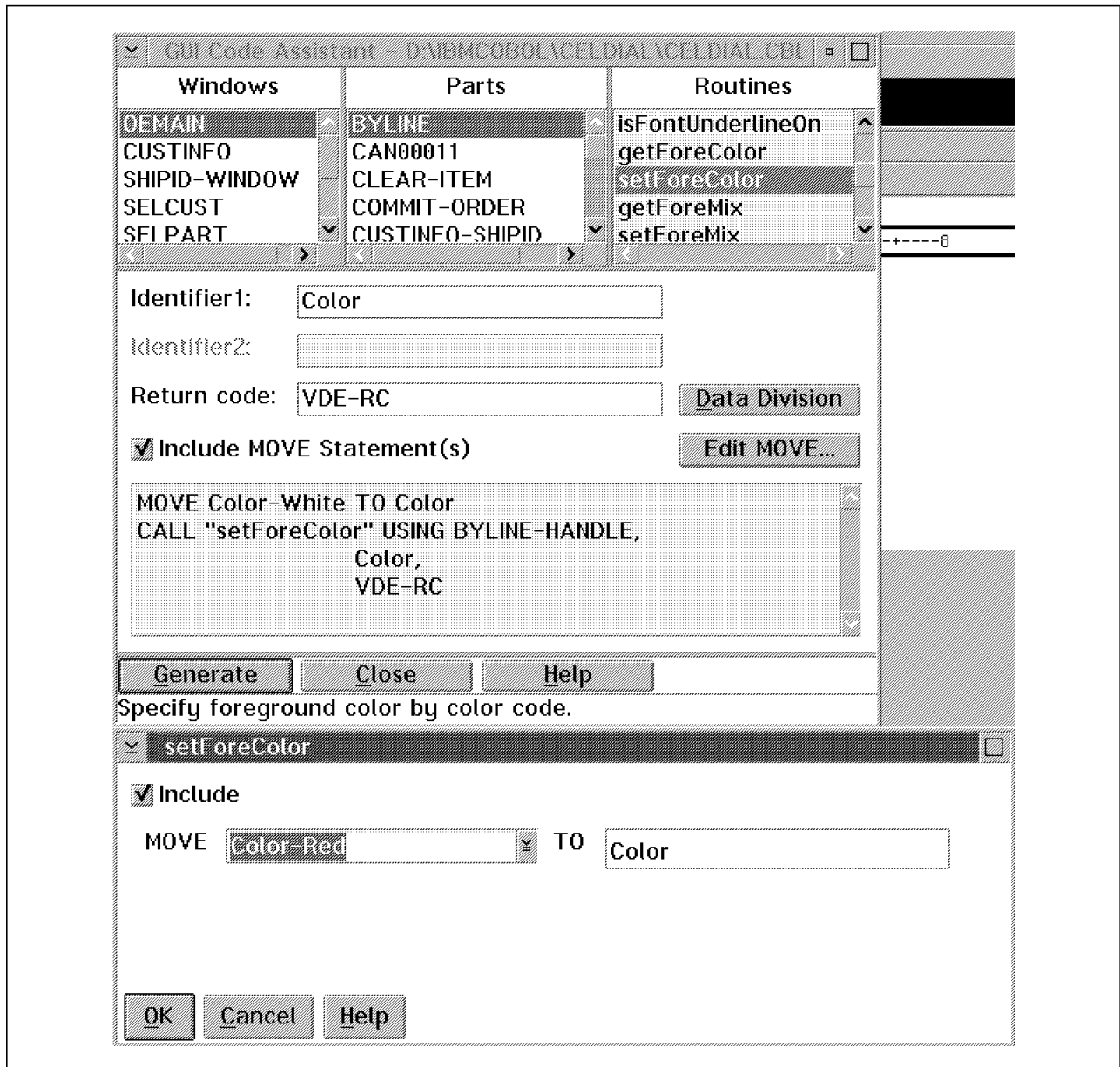


Figure 39. IBM VDE Code Assistant Window. Shows additions to change colors.

After carrying out this operation in our case, we added the following code to the event:

```

MOVE Color-Red TO Color
CALL "setForeColor" USING BYLINE-HANDLE,
                          Color,
                          VDE-RC

MOVE Color-White TO Color
CALL "setBackColor" USING BYLINE-HANDLE,
                          Color,
                          VDE-RC

```

The last statement has been migrated into this command,

```

CALL "enableFocus" USING OEMAIN-HANDLE,
                          VDE-RC

```

although it can be avoided. We decided to add it for illustrative purposes.

Now, starting from the beginning of the .TXT file, stop at each object that has a dialog to migrate into COBOL, working your way through the whole file.

7.1.1 Events for the OEMAIN-CUSTNO Entry Field

The following is the script for the events of this object

```
Dialog
  Event CR
    IF= " " CUSTNUM BLANK-CUSTNO ;
    MOVE 1 CUSTID-USERFILLED(1) ;
    RETC ;
    BRANCH-TO-PROCEDURE BLANK-CUSTNO ;
  End Event # CR
  Procedure BLANK-CUSTNO
    MOVE 1 CUSTLIST-BOTTON(1) ;
    RETC ;
    SET-DATA-GROUP-SIZE CUSTLIST CUST-SIZE ;
    SHOW-WINDOW SELCUST ;
    SET-FOCUS SELCUST ;
  End Procedure # VALID-CUSTNO
  Event GAINED-FOCUS
    MOVE "CR for customers" MSG-AREA ;
    REFRESH-OBJECT OEMAIN ;
  End Event # GAINED-FOCUS
  Event LOST-FOCUS
    MOVE " " MSG-AREA ;
    REFRESH-OBJECT OEMAIN ;
  End Event # LOST-FOCUS
End Dialog
```

The CR stands for the Carriage Return event (Figure 40), which has its VDE equivalent in the VDE VKEY-PRESS event when you press the Return (Newline) key. Open the VKEY-PRESS event and add a test to ensure that the Return key was pressed. The test is based on the variable VDE-VKey-Pressed-Data, defined as:

```
01 VDE-VKey-Pressed-Data.
05 VCharacter-Key          PIC 9(9) USAGE COMP-5.
   88 VKey-Escape          VALUE 0.
   88 VKey-Tab              VALUE 1.
   .....
   88 VKey-Newline          VALUE 6.
   .....
   88 VKey-F24              VALUE 47.
```

The code to add is:

```
ENTRY "OEMAIN_OEMAIN-CUSTNO_VKEYPRESS"
  USING VDE-CELDIAL,
        VDE-VKey-Pressed-Data.
IF VKey-Newline THEN

***** LOGIC *****

END-IF
GOBACK.
```

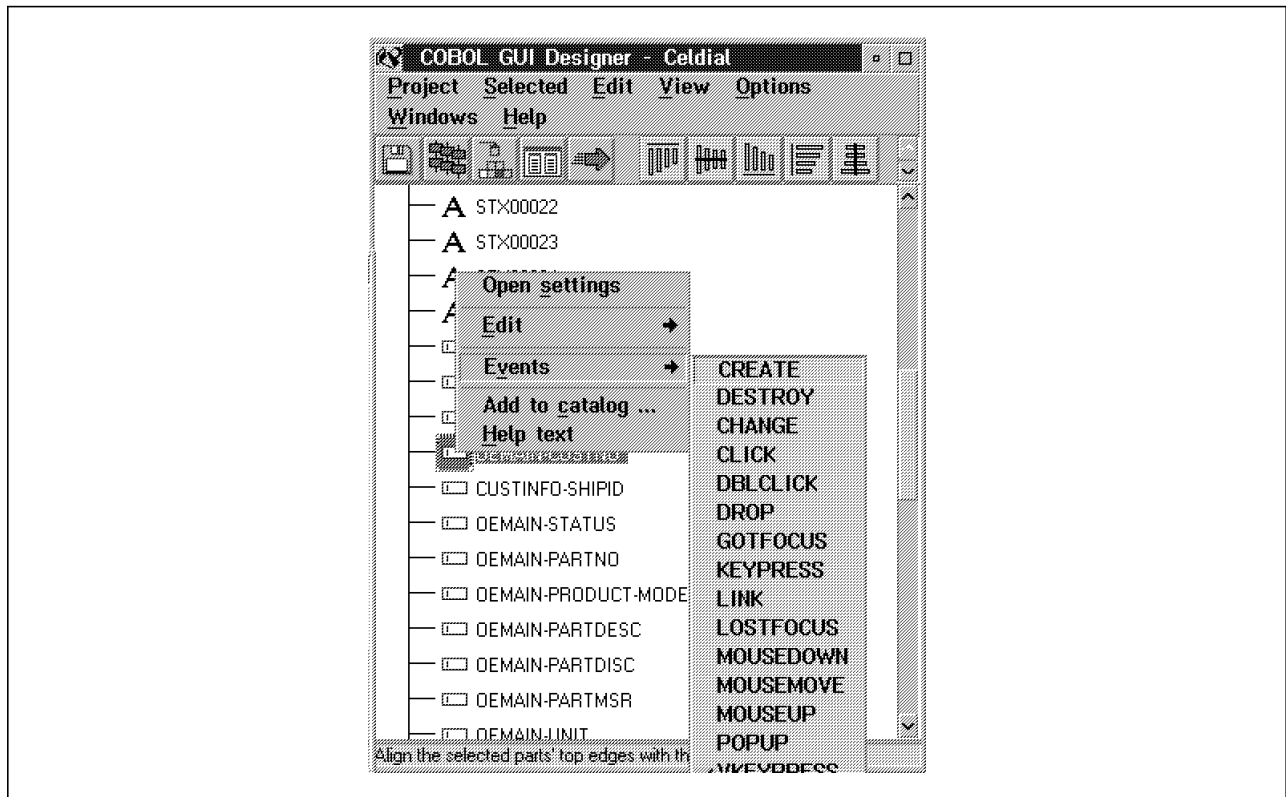


Figure 40. COBOL GUI Designer - Celdial Window. Shows additions for the carriage return event.

In Figure 40, use the tree-view approach to select the event. This is a valid way to select events when the window is too populated, since it takes less time to get the event skeleton code.

Now add the logic for the test on CUSTNUM:

```
IF CUSTNUM = " " THEN perform BLANK-CUSTNO END-IF
```

Then pass the control to the COBOL program, since we want to execute the routine associated with the flag "CUSTID-USERFILLED" Add the following statements:

```
move 1 to custid-userfilled
perform OLDMAIN
```

where OLDMAIN is the old COBOL program in our example. We import OLDMAIN because we want to be able to invoke all the sections it contains.

If the COBOL code has a variable linked to an object, MF Dialog System automatically refreshes the related object when the parent window is refreshed. In this case, the *ds-customer*, *ds-custno*, and the *custnum* values have been set.

Thus, to set the *customer number* entry field, add this logic:

```
MOVE length of ds-custno TO Contents-Length
MOVE ds-custno TO Contents-String
CALL "setContents" USING OEMAIN-CUSTNO-HANDLE,
                        Contents,
                        VDE-RC
```

The *customer:* variable updates the customer information automatically, just after creating and refreshing it.

Then, at the end, the event logic has this shape:

```
IF CUSTNUM = " " THEN perform BLANK-CUSTNO END-IF
  move 1 to custid-userfilled
  perform OLDMAIN
  perform BLANK-CUSTNO
```

You need to translate the BLANK-CUSTNO procedure into COBOL. Here is the result:

```
BLANK-CUSTNO section.
  MOVE 1 TO CUSTLIST-BOTTON
  PERFORM OLDMAIN
  CALL "openWindow" USING SELCUST-HANDLE,
                        VDE-RC

  MOVE 0 TO ItemIndex
  MOVE 0 TO counter
  perform CUST-SIZE times
    ADD 1 TO ItemIndex
    ADD 1 TO counter
    MOVE length of company(counter) TO Item-Length
    MOVE company(counter) TO Item-String
    CALL "insertItem" USING CUSTLIST-LIST-HANDLE,
                          ItemIndex,
                          Item,
                          VDE-RC
  end-perform
  CALL "enableFocus" USING SELCUST-HANDLE,
                        VDE-RC
```

In our example, we changed the logical order of certain statements. For example, we must open the window first and then populate it. If not, we would get an empty list box because we cannot populate the list box before it exists. If we test the VDE-RC in that case, we get a return code of 9, meaning that the object referred to in the `insertItem` statement is not valid (it does not yet exist).

The statement

```
SET-DATA-GROUP-SIZE CUSTLIST CUST-SIZE
```

has been translated into the loop

```
perform CUST-SIZE times ..... end-perform
```

The GAINED-FOCUS and LOST-FOCUS events became the events GOTFOCUS and LOSTFOCUS, respectively. After some work with the GUI assistant, we should get:

```
ENTRY "OEMAIN_OEMAIN-CUSTNO_GOTFOCUS"
  USING VDE-CELDIAL.
  MOVE "CR for customers" TO MSG-AREA

  MOVE length of MSG-AREA TO Contents-Length
  MOVE MSG-AREA TO Contents-String
  CALL "setContentts" USING MSG-TO-USER-HANDLE,
                          Contents,
                          VDE-RC
```

```

GOBACK.
ENTRY "OEMAIN_OEMAIN-CUSTNO_LOSTFOCUS"
    USING VDE-CELDIAL.
    MOVE " " TO MSG-AREA
    MOVE length of MSG-AREA TO Contents-Length
    MOVE MSG-AREA TO Contents-String
    CALL "setContents" USING MSG-TO-USER-HANDLE,
        Contents,
        VDE-RC
GOBACK.

```

7.1.2 Events for the CUSTINFO-SHIPID Entry Field

Events for this entry field are similar to events we discuss in Section 7.1.3.

7.1.3 Events for the OEMAIN-PARTNO Entry Field

This entry field contains four events to set up:

- Press **F9**
- Press **Enter** (NewLine)
- GotFocus
- LostFocus

For the first two events, use the VKEYPRESS event:

```

ENTRY "OEMAIN_OEMAIN-PARTNO_VKEYPRESS"
    USING VDE-CELDIAL,
        VDE-VKey-Pressed-Data.
IF VKey-F9 THEN perform PART-LISTBOX END-IF
IF VKey-Newline THEN
    IF COPART = " " THEN perform PART-LISTBOX END-IF
    move 1 to part-userfilled
    perform OLDMAIN
    perform PART-LISTBOX
END-IF.
PART-LISTBOX.
    MOVE 1 TO PART-FROM-LB
    perform OLDMAIN
    CALL "openWindow" USING SELPART-HANDLE,
        VDE-RC

    MOVE 0 TO ItemIndex
    MOVE 0 TO counter
    perform PROD-SIZE times
        ADD 1 TO ItemIndex
        ADD 1 TO counter
    MOVE 27 TO Item-Length
    MOVE spaces TO Item-String
    MOVE ds-partno(counter) TO Item-String(1:5)
    MOVE ds-partdesc(counter) TO Item-String(8:27)
    CALL "insertItem" USING PARTS-HANDLE,
        ItemIndex,
        Item,
        VDE-RC

    end-perform
    CALL "enableFocus" USING SELPART-HANDLE,
        VDE-RC
GOBACK.

```

7.1.4 Events for the OEMAIN-PRODUC-MODEL Entry Field

Two new statements are to be found among the events of this object:

```
CLEAR-OBJECT MODEL-LIST
CLEAR-OBJECT SELMODEL
```

In our case, these statements need not be converted because when we open a window, clearance is automatic. If the window is already open and you want to clear just the MODEL-LIST list box, add a command like this:

```
CALL "getCount" USING MODEL-LIST-HANDLE,
                    ItemCount,
                    VDE-RC
COMPUTE ItemIndex = ItemCount + 1
perform ItemCount times
    COMPUTE ItemIndex = ItemIndex -1
    CALL "removeItem" USING MODEL-LIST-HANDLE,
                        ItemIndex,
                        VDE-RC
end-perform
```

The remaining part of the migration for events of this object is trivial.

7.1.5 Events for the OEMAIN-UNIT Entry Field

The event CR (Newline) seems to be easy and straightforward but, for that very reason, look carefully at what happens in the COBOL program if there is a REFRESH command. Masterfields have probably been modified so that after the refresh action, the window content shows some modification. If you look for the section that the GO-COMPUTE flag launches, you find the following routine:

```
compute-subtot section.
    move 0 to go-compute
    compute cosubtotal = counit * coprice.
    compute total = total + cosubtotal.
```

Since the variables *COSUBTOTAL* and *TOTAL* have been modified, check to see what object they are related to. Get the content of the variables *counit* and *coprice* because if they have been changed from the application, those changes are reflected automatically in the related variables.

Objects involved in this routine are:

- OEMAIN-SUBTOTAL
- OEMAIN-TOTAL
- OEMAIN-UNIT-HANDLE
- OEMAIN-ORDER-DETAIL-P-HANDLE

Here is the logic we added for our example:

```
ENTRY "OEMAIN_OEMAIN-UNIT_VKEYPRESS"
    USING VDE-CELDIAL,
    VDE-VKey-Pressed-Data.

IF VKey-F9 THEN
    CALL "getContents" USING OEMAIN-UNIT-HANDLE,
                        Contents,
```

```

                                VDE-RC
MOVE Contents-String TO counit
CALL "getContents" USING OEMAIN-ORDER-DETAIL-P-HANDLE,
                                Contents,
                                VDE-RC
MOVE Contents-String TO coprice
MOVE 1 TO go-compute
perform OLDMAIN
MOVE length of cosubtotal TO Contents-Length
MOVE cosubtotal TO Contents-String
CALL "setContents" USING OEMAIN-SUBTAOTAL-HANDLE,
                                Contents,
                                VDE-RC
MOVE length of total TO Contents-Length
MOVE total TO Contents-String
CALL "setContents" USING OEMAIN-TOTAL-HANDLE,
                                Contents,
                                VDE-RC
END-IF
GOBACK.

```

The remaining events are straightforward and require no comment.

7.1.6 Events for the OEMAIN-CUSTINFO Push Button

Migration of the event looks easy, but unfortunately the logic that must be added has more statements than the logic to be migrated.

The original logic is as follows:

```

Dialog
  Event BUTTON-SELECTED
    IF= " " CUSTNUM BAD-CUSTNO ;
    MOVE 1 CUSTINFO-BOTTON(1) ;
    RETC ;
    SHOW-WINDOW CUSTINFO ;
    SET-FOCUS CUSTINFO ;
    REFRESH-OBJECT CUSTINFO ;
  End Event # BUTTON-SELECTED
  Procedure BAD-CUSTNO
    INVOKE-MESSAGE-BOX NO-CUSTNO $NULL $REGISTER ;
  End Procedure # BAD-CUSTNO
End Dialog

```

The **IF** statement checks whether the customer number is blank, but to do so, you need the content of that field. You then need to add the routine to invoke the message box.

First, you need to know which entry field is related to the *custnum*, variable, then get its value, and finally test if its content is empty. If so, you need to invoke a routine that shows the message box. Since there is no link between the Dialog System message box name and the VDE message box number, you must also look for the correct message in the VDE message list. Once you have the message number, we are ready for the migration of the whole event.

Here is the logic you need:

```

ENTRY "OEMAIN_OEMAIN-CUSTINFO_PRESS"
    USING VDE-CELDIAL.
    CALL "getContents" USING OEMAIN-CUSTNO-HANDLE,
                                Contents,
                                VDE-RC
    MOVE Contents-String TO CUSTNUM
    IF CUSTNUM = " " THEN perform BAD-CUSTNO
    ELSE
        .....
        .....
    END-IF
    GOBACK.
bad-custno.
    MOVE 2 TO MessageNumber
    MOVE 0 TO NumberOfSubStrings
    CALL "showMsgFromMsgFile" USING MsgFileRecord,
                                MessageButtonReturned,
                                VDE-RC.

end-bad-custno.

```

For the rest of the event, you need only get and set contents. There is nothing of interest in these changes.

7.1.7 Events for the ORDEROK Radio Button

The MF event for this object is :

```

Dialog
    Event BUTTON-SELECTED
        IF= 0 COSUBTOTAL(1) DONT-SAVE ;
        MOVE 1 ITEM-OK-BOTTON(1) ;
        RETC ;
        CLEAR-OBJECT ORDEROK ;
        SET-DATA-GROUP-SIZE SAVORDER SAVORDER-SIZE ;
        REFRESH-OBJECT OEMAIN-ORDERS ;
        REFRESH-OBJECT OEMAIN ;
    End Event # BUTTON-SELECTED
    Procedure DONT-SAVE
        End Procedure # DONT-SAVE
    End Dialog

```

The only item of interest in this event is the CLEAR-OBJECT command. In this case, since the object is a radio button, this means setting the radio button to OFF. The equivalent in VDE is:

```

CALL "disableCheck" USING ORDEROK-HANDLE,
                                VDE-RC

```

The procedure you execute when you pass the ITEM-OK-BOTTON flag to the COBOL program contains an INITIALIZE statement. This means that all of the variables and the entry fields related to those variables are updated after a REFRESH command.

7.1.8 Events for the OEMAIN-NEWORDER Push Button

The MF event code is as follows:

```
Dialog
  Event BUTTON-SELECTED
    CLEAR-OBJECT OEMAIN-ORDERS ;
    CLEAR-OBJECT OEMAIN ;
    MOVE 1 NEWORDER-BOTTON(1) ;
    RETC ;
    SET-DATA-GROUP-SIZE SAVORDER 0 ;
    REFRESH-OBJECT OEMAIN-ORDERS ;
    REFRESH-OBJECT OEMAIN ;
    SET-FOCUS OEMAIN ;
  End Event # BUTTON-SELECTED
End Dialog
```

The first statement in the button-selected event code says you need to clear the list box containing orders. First get the number of the items in the list box and then delete them one by one. Here is the code you need to add for such a job:

```
ENTRY "OEMAIN_OEMAIN-NEWORDER_PRESS"
  USING VDE-CELDIAL.
CALL "getCount" USING OEMAIN-ORDERS-HANDLE,
                    ItemCount,
                    VDE-RC

MOVE ItemCount TO counter
MOVE 1 TO ItemIndex
perform counter times
CALL "removeItem" USING OEMAIN-ORDERS-HANDLE,
                    ItemIndex,
                    VDE-RC

end-perform
.....
```

Look also at the content of the perform statement you execute in the OLDMAIN procedure. You need to update the entry field related to the *ordernum* variable.

7.1.9 Events for the REVISE-ORDER Radio Button

We can find something new in the Micro Focus event for this object:

```
Dialog
  Event BUTTON-SELECTED
    MOVE 1 REVISE-ORDER-BOTTON(1) ;
    GET-SELECTED-LIST-ITEM OEMAIN-ORDERS SELECTED-SAVEDITEM $EVENT-DATA ;
    MOVE $EVENT-DATA SELECTED-SAVEDITEM ;
    RETC ;
    CLEAR-OBJECT REVISE-ORDER ;
    DELETE-LIST-ITEM OEMAIN-ORDERS SELECTED-SAVEDITEM $EVENT-DATA ;
    SET-DATA-GROUP-SIZE SAVORDER SAVORDER-SIZE ;
    REFRESH-OBJECT OEMAIN ;
  End Event # BUTTON-SELECTED
End Dialog
```

Statements that get the selected list item and delete it translate almost one to one, but their sequence is not the same. In fact, in the Dialog System environment, to delete a row for a list box, do it after the RETC command, in order to use new values and then refresh the window with them. In the VDE environment, you can delete the item before the RETC command because there is no link between variables and objects. The result of the migration is:

```

ENTRY "OEMAIN_REVERSE-ORDER_SELECT"
    USING VDE-CELDIAL.
MOVE 1 TO revise-order-button
CALL "getFirstSelected" USING OEMAIN-ORDERS-HANDLE,
    ItemIndex,
    VDE-RC
MOVE ItemIndex TO Selected-saveditem
CALL "removeItem" USING OEMAIN-ORDERS-HANDLE,
    ItemIndex,
    VDE-RC

perform OLDMAIN

MOVE length of total TO Contents-Length
MOVE total TO Contents-String
CALL "setContentts" USING OEMAIN-TOTAL-HANDLE,
    Contents,
    VDE-RC

.....

CALL "disableCheck" USING REVISE-ORDER-HANDLE,
    VDE-RC
GOBACK.

```

Chapter 8. Installing the Migration Utility

8.1 Hardware and Software Prerequisites

Every machine that can run VisualAge for COBOL has all of the prerequisites this utility needs. The only feature to check is whether the REXX environment was installed during the system installation. If not, you have to install it in order to execute this utility.

The utility needs a total space of about 2 MB.

8.2 Installing the Migration Utility

To install the Migration Utility, carry out these steps:

1. Create a directory (we created the MF2VDE directory).
2. From the diskette, copy into the directory the MF2VDE.ZIP file. Unpack it with the command:

```
PKUNZIP MF2VDE.ZIP
```

Here is the list of files you should have:

DATAVAL.CMD	VROBJ.DLL
CELDIAL.TXT	ODGUIDOF
KEYS	MB
KEYS.CMD	MF2VDE.ICO
STYLES.CMD	30.SCR
ATTRIBUT.CMD	30E.SCR
LOGFILE	TEMPFILE
LOGWRITE.CMD	MF2VDEPS.CMD
IWZVMKG.EXE	WINDOW
CHKLIST.MS	EF
CELDIAL.BC	TEXT
MF2VDE.ERR	TRIAL001
MIGRATIO.NEW	MLE
MIGRATIO.BC	PB
MIGRATIO.P	RB
RMKG.CMD	CB
BITMAP.CMD	CHB
CB.CMD	MENU
CHB.CMD	XOR.CMD
EF.CMD	IWZVCC.DLL
GB.CMD	MIGRATIO.TXT
LB.CMD	LB
MB.CMD	GB
MENU.CMD	BITMAP
MLE.CMD	BOTCORN.CMD
PB.CMD	SIZECONV.CMD
RB.CMD	PRVA.CMD
TEXT.CMD	M2VLABEL.CMD
WINDOW.CMD	STYLENAM.CMD
MF2VDE.VRX	CATALOG.CMD
MF2VDE.VRP	GPB
MF2VDE.VRY	EF.OLD

MF2VDE.EXE	DATATYP.CMD
A.A	PMWIN.H

3. Edit the RMKG.CMD file and change the value of the *somir* variable by changing its path. For example:
 from the value : set somir=d:\ibmcobol\ctrl\iwzvsom.ir
 to the value : set somir=C:\MYcobol\ctrl\iwzvsom.ir
 Do not change the \ctrl\iwzvsom.ir string.
4. Copy the RMKG.CMD and the IWZVMKG.EXE files into the \IBMCOBOL\BIN directory.
5. Create the program icon on the desktop using the PROGRAM template from the TEMPLATE folder. Drag it, drop it to the desktop and specify path name, program name, and working directory as in Figure 41. You may not have VXREXX as the root of your working directory.
6. Make sure that in your CONFIG.SYS, in the SET LIBPATH command, the current directory string is specified.

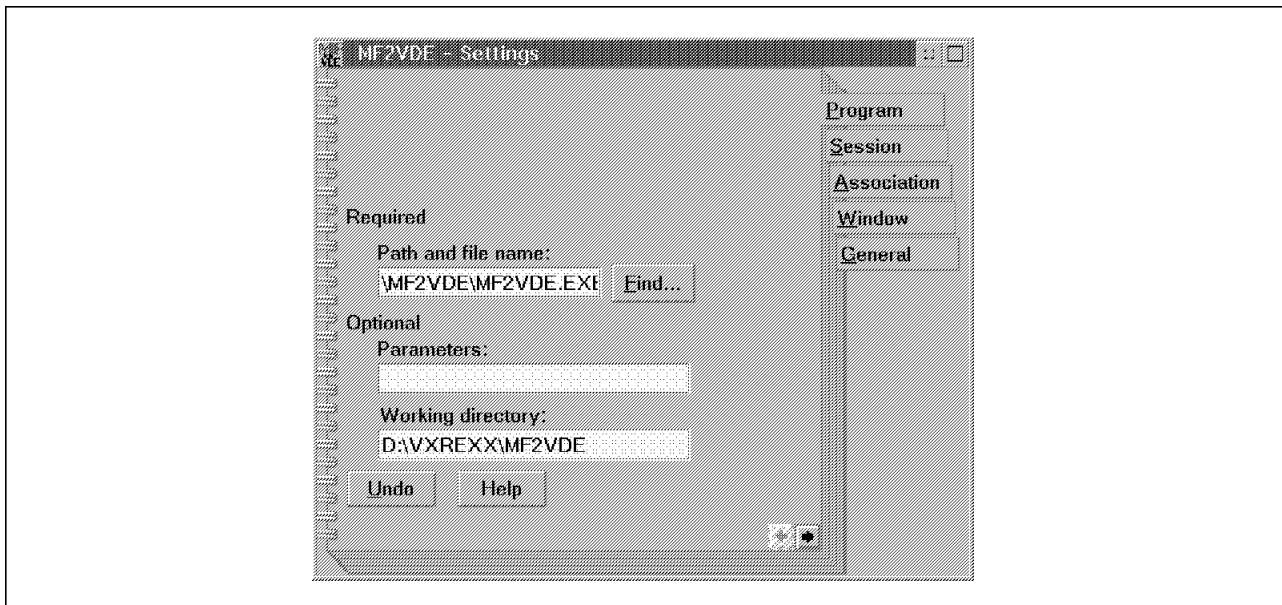


Figure 41. MF2VDE - Settings Window Showing the Migration Utility Settings

If you should need to change the utility logic, you may need to use the VX-REXX product. Logic for parsing the objects is included in the .CMD file, which can be changed without the VX-REXX product. The objects are pure REXX code; they use some of the VX-REXX functions but they have not been compiled and included in the MF2VDE.EXE file.

The main source file is MF2VDE.VRX.

You cannot successfully modify this source if you do not use the VX-REXX product.

Appendix A. Special Notices

This publication is intended to help application developers to investigate how to migrate Micro Focus Dialog System OS/2 applications to IBM VisualAge for COBOL for OS/2. The information in this publication is not intended as the specification of any programming interfaces that are provided by IBM VisualAge for COBOL for OS/2. See the PUBLICATIONS section of the IBM Programming Announcement for IBM VisualAge for COBOL for OS/2 for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM ("vendor") products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

DATABASE 2	DB2
DB2/2	IBM
PS/2	VisualAge

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Microsoft, Windows, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

Java and HotJava are trademarks of Sun Microsystems, Inc.

Micro Focus and Dialog System are trademarks of Micro Focus Corporation

VX-REXX is a trademark of Powersoft, Watcom Products Division.

Powersoft is a trademark of Powersoft Corporation.

Other trademarks are trademarks of their respective companies.

Appendix B. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

B.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see "How to Get ITSO Redbooks" on page 63.

- *IBM VisualAge for COBOL for OS/2 Primer*, SG24-4605
- *IBM VisualAge for COBOL for OS/2 Workframe User Guide*, SG24-4604
- *IBM VisualAge for COBOL for OS/2 Object Oriented Programming*, SG24-4606

B.2 Redbooks on CD-ROMs

Redbooks are also available on CD-ROMs. **Order a subscription** and receive updates 2-4 times a year at significant savings.

CD-ROM Title	Subscription Number	Collection Kit Number
System/390 Redbooks Collection	SBOF-7201	SK2T-2177
Networking and Systems Management Redbooks Collection	SBOF-7370	SK2T-6022
Transaction Processing and Data Management Redbook	SBOF-7240	SK2T-8038
AS/400 Redbooks Collection	SBOF-7270	SK2T-2849
RS/6000 Redbooks Collection (HTML, BkMgr)	SBOF-7230	SK2T-8040
RS/6000 Redbooks Collection (PostScript)	SBOF-7205	SK2T-8041
Application Development Redbooks Collection	SBOF-7290	SK2T-8037
Personal Systems Redbooks Collection	SBOF-7250	SK2T-8042

B.3 Other Publications

These publications are also relevant as further information sources:

- *IBM VisualAge for COBOL for OS/2 Language Reference*, SC26-4769
- *IBM VisualAge for COBOL for OS/2 Programming Guide*, SC26-8419

How to Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, CD-ROMs, workshops, and residencies. A form for ordering books and CD-ROMs is also provided.

This information was current at the time of publication, but is continually subject to change. The latest information may be found at URL <http://www.redbooks.ibm.com>.

How IBM Employees Can Get ITSO Redbooks

Employees may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **PUBORDER** — to order hardcopies in United States
- **GOPHER link to the Internet** - type GOPHER.WTSCPOK.ITSO.IBM.COM
- **Tools disks**

To get LIST3820s of redbooks, type one of the following commands:

```
TOOLS SENDTO EHONE4 TOOLS2 REDPRINT GET SG24xxxx PACKAGE
TOOLS SENDTO CANVM2 TOOLS REDPRINT GET SG24xxxx PACKAGE (Canadian users only)
```

To get BookManager BOOKs of redbooks, type the following command:

```
TOOLCAT REDBOOKS
```

To get lists of redbooks:

```
TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET ITSOCAT TXT
TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET LISTSERV PACKAGE
```

To register for information on workshops, residencies, and redbooks:

```
TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1996
```

For a list of product area specialists in the ITSO:

```
TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ORGCARD PACKAGE
```

- **Redbooks Home Page on the World Wide Web**
<http://w3.itso.ibm.com/redbooks>
- **IBM Direct Publications Catalog on the World Wide Web**
<http://www.elink.ibm.link.ibm.com/pbl/pbl>

IBM employees may obtain LIST3820s of redbooks from this page.

- **REDBOOKS category on INEWS**
- **Online** — send orders to: USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL
- **Internet Listserver**

With an Internet e-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an e-mail note to announce@webster.ibm.link.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank). A category form and detailed instructions will be sent to you.

How Customers Can Get ITSO Redbooks

Customers may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Online Orders** (Do not send credit card information over the Internet) — send orders to:

	IBMMAIL	Internet
In United States:	usib6fpl at ibmmail	usib6fpl@ibmmail.com
In Canada:	caibmbkz at ibmmail	lmannix@vnet.ibm.com
Outside North America:	dkibmbsh at ibmmail	bookshop@dk.ibm.com

- **Telephone orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	(long distance charges apply)
(+45) 4810-1320 - Danish	(+45) 4810-1020 - German
(+45) 4810-1420 - Dutch	(+45) 4810-1620 - Italian
(+45) 4810-1540 - English	(+45) 4810-1270 - Norwegian
(+45) 4810-1670 - Finnish	(+45) 4810-1120 - Spanish
(+45) 4810-1220 - French	(+45) 4810-1170 - Swedish

- **Mail Orders** — send orders to:

IBM Publications Publications Customer Support P.O. Box 29570 Raleigh, NC 27626-0570 USA	IBM Publications 144-4th Avenue, S.W. Calgary, Alberta T2P 3N5 Canada	IBM Direct Services Sortemosevej 21 DK-3450 Allerød Denmark
--	--	--

- **Fax** — send orders to:

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	(+45) 48 14 2207 (long distance charge)

- **1-800-IBM-4FAX (United States)** or **(+1)001-408-256-5422 (Outside USA)** — ask for:

Index # 4421 Abstracts of new redbooks
Index # 4422 IBM redbooks
Index # 4420 Redbooks for last six months

- **Direct Services** - send note to softwareshop@vnet.ibm.com

- **On the World Wide Web**

Redbooks Home Page	http://www.redbooks.ibm.com
IBM Direct Publications Catalog	http://www.elink.ibm.link.ibm.com/pbl/pbl

- **Internet Listserver**

With an Internet e-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an e-mail note to announce@webster.ibm.link.ibm.com with the keyword `subscribe` in the body of the note (leave the subject line blank).

IBM Redbook Order Form

Please send me the following:

Title	Order Number	Quantity

First name	Last name
------------	-----------

Company

Address

City	Postal code	Country
------	-------------	---------

Telephone number	Telefax number	VAT number
------------------	----------------	------------

• Invoice to customer number _____

• Credit card number _____

Credit card expiration date	Card issued to	Signature
-----------------------------	----------------	-----------

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

DO NOT SEND CREDIT CARD INFORMATION OVER THE INTERNET.

Glossary

The terms in this glossary are defined in accordance with their meaning in COBOL. These terms may or may not have the same meaning in other languages.

IBM is grateful to the American National Standards Institute (ANSI) for permission to reprint its definitions from the following publications:

- *American National Standard Programming Language COBOL, ANSI X3.23-1985* (Copyright 1985 American National Standards Institute, Inc.), which was prepared by Technical Committee X3J4, which had the task of revising American National Standard COBOL, X3.23-1974.
- *American National Dictionary for Information Processing Systems* (Copyright 1982 by the Computer and Business Equipment Manufacturers Association).

American National Standard definitions are preceded by an asterisk (*).

A

*** abbreviated combined relation condition.** The combined condition that results from the explicit omission of a common subject or a common subject and common relational operator in a consecutive sequence of relation conditions.

abend. Abnormal termination of program.

*** access mode.** The manner in which records are to be operated upon within a file.

*** actual decimal point.** The physical representation, using the decimal point characters period (.) or comma (,), of the decimal point position in a data item.

*** alphabet-name.** A user-defined word, in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION, that assigns a name to a specific character set and/or collating sequence.

*** alphabetic character.** A letter or a space character.

*** alphanumeric character.** Any character in the computer's character set.

alphanumeric-edited character. A character within an alphanumeric character-string that contains at least one B, 0 (zero), or / (slash).

*** alphanumeric function.** A function whose value is composed of a string of one or more characters from the computer's character set.

*** alternate record key.** A key, other than the prime record key, whose contents identify a record within an indexed file.

ANSI (American National Standards Institute). An organization consisting of producers, consumers, and general interest groups, that establishes the procedures by which accredited organizations create and maintain voluntary industry standards in the United States.

*** argument.** An identifier, a literal, an arithmetic expression, or a function-identifier that specifies a value to be used in the evaluation of a function.

*** arithmetic expression.** An identifier of a numeric elementary item, a numeric literal, such identifiers and literals separated by arithmetic operators, two arithmetic expressions separated by an arithmetic operator, or an arithmetic expression enclosed in parentheses.

*** arithmetic operation.** The process caused by the execution of an arithmetic statement, or the evaluation of an arithmetic expression, that results in a mathematically correct solution to the arguments presented.

*** arithmetic operator.** A single character, or a fixed two-character combination that belongs to the following set:

Character	Meaning
+	addition
—	subtraction
*	multiplication
/	division
**	exponentiation

*** arithmetic statement.** A statement that causes an arithmetic operation to be executed. The arithmetic statements are the ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT statements.

array. In Language Environment, an aggregate consisting of data objects, each of which may be uniquely referenced by subscripting. Roughly analogous to a COBOL table.

*** ascending key.** A key upon the values of which data is ordered, starting with the lowest value of the key up to the highest value of the key, in accordance with the rules for comparing data items.

ASCII. American National Standard Code for Information Interchange. The standard code, using a coded character set consisting of 7-bit coded characters (8 bits including parity check), used for information interchange between data processing systems, data communication systems, and

associated equipment. The ASCII set consists of control characters and graphic characters.

Note: IBM has defined an extension to ASCII code (characters 128-255).

assignment-name. A name that identifies the organization of a COBOL file and the name by which it is known to the system.

*** assumed decimal point.** A decimal point position that does not involve the existence of an actual character in a data item. The assumed decimal point has logical meaning with no physical representation.

*** AT END condition.** A condition caused:

1. During the execution of a READ statement for a sequentially accessed file, when no next logical record exists in the file, or when the number of significant digits in the relative record number is larger than the size of the relative key data item, or when an optional input file is not present.
2. During the execution of a RETURN statement, when no next logical record exists for the associated sort or merge file.
3. During the execution of a SEARCH statement, when the search operation terminates without satisfying the condition specified in any of the associated WHEN phrases.

B

big-endian. Default format used by the mainframe and the AIX workstation to store binary data. In this format, the least significant digit is on the highest address. Compare with "little-endian."

binary item. A numeric data item represented in binary notation (on the base 2 numbering system). Binary items have a decimal equivalent consisting of the decimal digits 0 through 9, plus an operational sign. The leftmost bit of the item is the operational sign.

binary search. A dichotomizing search in which, at each step of the search, the set of data elements is divided by two; some appropriate action is taken in the case of an odd number.

*** block.** A physical unit of data that is normally composed of one or more logical records. For mass storage files, a block may contain a portion of a logical record. The size of a block has no direct relationship to the size of the file within which the block is contained or to the size of the logical record(s) that are either contained within the block or that overlap the block. The term is synonymous with physical record.

breakpoint. A place in a computer program, usually specified by an instruction, where its execution may

be interrupted by external intervention or by a monitor program.

Btrieve. A key-indexed record management system that allows applications to manage records by key value, sequential access method, or random access method. IBM COBOL supports COBOL sequential and indexed file I-O language through Btrieve.

buffer. A portion of storage used to hold input or output data temporarily.

built-in function. See "intrinsic function".

byte. A string consisting of a certain number of bits, usually eight, treated as a unit, and representing a character.

C

callable services. In Language Environment, a set of services that can be invoked by a COBOL program using the conventional Language Environment-defined call interface, and usable by all programs sharing the Language Environment conventions.

called program. A program that is the object of a CALL statement.

*** calling program.** A program that executes a CALL to another program.

case structure. A program processing logic in which a series of conditions is tested in order to make a choice between a number of resulting actions.

cataloged procedure. A set of job control statements placed in a partitioned data set called the procedure library (SYS1.PROCLIB). You can use cataloged procedures to save time and reduce errors coding JCL.

century window. The 100-year interval in which Language Environment assumes all 2-digit years lie. The Language Environment default century window begins 80 years before the system date.

*** character.** The basic indivisible unit of the language.

character position. The amount of physical storage required to store a single standard data format character described as USAGE IS DISPLAY.

character set. All the valid characters for a programming language or a computer system.

*** character-string.** A sequence of contiguous characters that form a COBOL word, a literal, a PICTURE character-string, or a comment-entry. Must be delimited by separators.

checkpoint. A point at which information about the status of a job and the system can be recorded so that the job step can be later restarted.

*** class.** The entity that defines common behavior and implementation for zero, one, or more objects. The objects that share the same implementation are considered to be objects of the same class.

*** class condition.** The proposition, for which a truth value can be determined, that the content of an item is wholly alphabetic, is wholly numeric, or consists exclusively of those characters listed in the definition of a class-name.

*** Class Definition.** The COBOL source unit that defines a class.

*** class identification entry.** An entry in the CLASS-ID paragraph of the IDENTIFICATION DIVISION which contains clauses that specify the class-name and assign selected attributes to the class definition.

*** class-name.** A user-defined word defined in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION that assigns a name to the proposition for which a truth value can be defined, that the content of a data item consists exclusively of those characters listed in the definition of the class-name.

class object. The run-time object representing a SOM class.

*** clause.** An ordered set of consecutive COBOL character-strings whose purpose is to specify an attribute of an entry.

CMS (Conversational Monitor System). A virtual machine operating system that provides general interactive, time-sharing, problem solving, and program development capabilities, and that operates only under the control of the VM/SP control program.

*** COBOL character set.** The complete COBOL character set consists of the characters listed below:

Character	Meaning
0,1,...,9	digit
A,B,...,Z	uppercase letter
a,b,...,z	lowercase letter
•	space
+	plus sign
—	minus sign (hyphen)
*	asterisk
/	slant (virgule, slash)
=	equal sign
\$	currency sign
,	comma (decimal point)
;	semicolon
.	period (decimal point, full stop)
”	quotation mark
(	left parenthesis

)	right parenthesis
>	greater than symbol
<	less than symbol
:	colon

*** COBOL word.** See “word.”

code page. An assignment of graphic characters and control function meanings to all code points; for example, assignment of characters and meanings to 256 code points for 8-bit code, assignment of characters and meanings to 128 code points for 7-bit code.

*** collating sequence.** The sequence in which the characters that are acceptable to a computer are ordered for purposes of sorting, merging, comparing, and for processing indexed files sequentially.

*** column.** A character position within a print line. The columns are numbered from 1, by 1, starting at the leftmost character position of the print line and extending to the rightmost position of the print line.

*** combined condition.** A condition that is the result of connecting two or more conditions with the AND or the OR logical operator.

*** comment-entry.** An entry in the IDENTIFICATION DIVISION that may be any combination of characters from the computer's character set.

*** comment line.** A source program line represented by an asterisk (*) in the indicator area of the line and any characters from the computer's character set in area A and area B of that line. The comment line serves only for documentation in a program. A special form of comment line represented by a slant (/) in the indicator area of the line and any characters from the computer's character set in area A and area B of that line causes page ejection prior to printing the comment.

*** common program.** A program which, despite being directly contained within another program, may be called from any program directly or indirectly contained in that other program.

*** compile.** (1) To translate a program expressed in a high-level language into a program expressed in an intermediate language, assembly language, or a computer language. (2) To prepare a machine language program from a computer program written in another programming language by making use of the overall logic structure of the program, or generating more than one computer instruction for each symbolic statement, or both, as well as performing the function of an assembler.

*** compile time.** The time at which a COBOL source program is translated, by a COBOL compiler, to a COBOL object program.

compiler. A program that translates a program written in a higher level language into a machine language object program.

compiler directing statement. A statement, beginning with a compiler directing verb, that causes the compiler to take a specific action during compilation.

compiler directing statement. A statement that specifies actions to be taken by the compiler during processing of a COBOL source program. Compiler directives are contained in the COBOL source program. Thus, you can specify different suboptions of the directive within the source program by using multiple compiler directive statements in the program.

*** complex condition.** A condition in which one or more logical operators act upon one or more conditions. (See also "negated simple condition," "combined condition," and "negated combined condition.")

*** computer-name.** A system-name that identifies the computer upon which the program is to be compiled or run.

condition. An exception that has been enabled, or recognized, by Language Environment and thus is eligible to activate user and language condition handlers. Any alteration to the normal programmed flow of an application. Conditions can be detected by the hardware/operating system and results in an interrupt. They can also be detected by language-specific generated code or language library code.

*** condition.** A status of a program at run time for which a truth value can be determined. Where the term 'condition' (condition-1, condition-2,...) appears in these language specifications in or in reference to 'condition' (condition-1, condition-2,...) of a general format, it is a conditional expression consisting of either a simple condition optionally parenthesized, or a combined condition consisting of the syntactically correct combination of simple conditions, logical operators, and parentheses, for which a truth value can be determined.

*** conditional expression.** A simple condition or a complex condition specified in an EVALUATE, IF, PERFORM, or SEARCH statement. (See also "simple condition" and "complex condition.")

*** conditional phrase.** A conditional phrase specifies the action to be taken upon determination of the truth value of a condition resulting from the execution of a conditional statement.

*** conditional statement.** A statement specifying that the truth value of a condition is to be determined and that the subsequent action of the object program is dependent on this truth value.

*** conditional variable.** A data item one or more values of which has a condition-name assigned to it.

*** condition-name.** A user-defined word that assigns a name to a subset of values that a conditional variable may assume; or a user-defined word assigned to a status of an implementor defined switch or device. When 'condition-name' is used in the general formats, it represents a unique data item reference consisting of a syntactically correct combination of a 'condition-name', together with qualifiers and subscripts, as required for uniqueness of reference.

*** condition-name condition.** The proposition, for which a truth value can be determined, that the value of a conditional variable is a member of the set of values attributed to a condition-name associated with the conditional variable.

*** CONFIGURATION SECTION.** A section of the ENVIRONMENT DIVISION that describes overall specifications of source and object programs and class definitions.

CONSOLE. A COBOL environment-name associated with the operator console.

*** contiguous items.** Items that are described by consecutive entries in the Data Division, and that bear a definite hierarchic relationship to each other.

CORBA. The Common Object Request Broker Architecture established by the Object Management Group. IBM's *Interface Definition Language* used to describe the *interface* for SOM classes is fully compliant with CORBA standards.

*** counter.** A data item used for storing numbers or number representations in a manner that permits these numbers to be increased or decreased by the value of another number, or to be changed or reset to zero or to an arbitrary positive or negative value.

cross-reference listing. The portion of the compiler listing that contains information on where files, fields, and indicators are defined, referenced, and modified in a program.

currency sign. The character '\$' of the COBOL character set or that character defined by the CURRENCY compiler option. If the NOCURRENCY compiler option is in effect, the currency sign is defined as the character '\$'.

currency symbol. The character defined by the CURRENCY compiler option or by the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph. If the NOCURRENCY compiler option is in effect for a COBOL source program and the CURRENCY SIGN clause is also **not** present in the source program, the currency symbol is identical to the currency sign.

* **current record.** In file processing, the record that is available in the record area associated with a file.

* **current volume pointer.** A conceptual entity that points to the current volume of a sequential file.

D

* **data clause.** A clause, appearing in a data description entry in the DATA DIVISION of a COBOL program, that provides information describing a particular attribute of a data item.

* **data description entry .** An entry in the DATA DIVISION of a COBOL program that is composed of a level-number followed by a data-name, if required, and then followed by a set of data clauses, as required.

DATA DIVISION. One of the four main components of a COBOL program, class definition, or method definition. The DATA DIVISION describes the data to be processed by the object program, class, or method: files to be used and the records contained within them; internal working-storage records that will be needed; data to be made available in more than one program in the COBOL run unit. (Note, the Class DATA DIVISION contains only the WORKING-STORAGE SECTION.)

* **data item.** A unit of data (excluding literals) defined by a COBOL program or by the rules for function evaluation.

* **data-name.** A user-defined word that names a data item described in a data description entry. When used in the general formats, 'data-name' represents a word that must not be reference-modified, subscripted or qualified unless specifically permitted by the rules for the format.

DBCS (Double-Byte Character Set). See "Double-Byte Character Set (DBCS)."

* **debugging line.** A debugging line is any line with a 'D' in the indicator area of the line.

* **debugging section.** A section that contains a USE FOR DEBUGGING statement.

* **declarative sentence.** A compiler directing sentence consisting of a single USE statement terminated by the separator period.

* **declaratives.** A set of one or more special purpose sections, written at the beginning of the Procedure Division, the first of which is preceded by the key word DECLARATIVES and the last of which is followed by the key words END DECLARATIVES. A declarative is composed of a section header, followed by a USE compiler directing sentence, followed by a set of zero, one, or more associated paragraphs.

* **de-edit.** The logical removal of all editing characters from a numeric edited data item in order to determine that item's unedited numeric value.

* **delimited scope statement.** Any statement that includes its explicit scope terminator.

* **delimiter.** A character or a sequence of contiguous characters that identify the end of a string of characters and separate that string of characters from the following string of characters. A delimiter is not part of the string of characters that it delimits.

* **descending key.** A key upon the values of which data is ordered starting with the highest value of key down to the lowest value of key, in accordance with the rules for comparing data items.

digit. Any of the numerals from 0 through 9. In COBOL, the term is not used in reference to any other symbol.

* **digit position.** The amount of physical storage required to store a single digit. This amount may vary depending on the usage specified in the data description entry that defines the data item.

* **direct access.** The facility to obtain data from storage devices or to enter data into a storage device in such a way that the process depends only on the location of that data and not on a reference to data previously accessed.

* **division.** A collection of zero, one or more sections or paragraphs, called the division body, that are formed and combined in accordance with a specific set of rules. Each division consists of the division header and the related division body. There are four (4) divisions in a COBOL program: Identification, Environment, Data, and Procedure.

* **division header.** A combination of words followed by a separator period that indicates the beginning of a division. The division headers are:

IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION.

do construction. In structured programming, a DO statement is used to group a number of statements in a procedure. In COBOL, an in-line PERFORM statement functions in the same way.

do-until. In structured programming, a do-until loop will be executed at least once, and until a given condition is true. In COBOL, a TEST AFTER phrase used with the PERFORM statement functions in the same way.

do-while. In structured programming, a do-while loop will be executed if, and while, a given condition is

true. In COBOL, a TEST BEFORE phrase used with the PERFORM statement functions in the same way.

Double-Byte Character Set (DBCS). A set of characters in which each character is represented by two bytes. Languages such as Japanese, Chinese, and Korean, which contain more symbols than can be represented by 256 code points, require Double-Byte Character Sets. Because each character requires two bytes, entering, displaying, and printing DBCS characters requires hardware and supporting software that are DBCS-capable.

*** dynamic access.** An access mode in which specific logical records can be obtained from or placed into a mass storage file in a nonsequential manner and obtained from a file in a sequential manner during the scope of the same OPEN statement.

Dynamic Storage Area (DSA). Dynamically acquired storage composed of a register save area and an area available for dynamic storage allocation (such as program variables). DSAs are generally allocated within STACK segments managed by Language Environment.

E

*** EBCDIC (Extended Binary-Coded Decimal Interchange Code).** A coded character set consisting of 8-bit coded characters.

EBCDIC character. Any one of the symbols included in the 8-bit EBCDIC (Extended Binary-Coded-Decimal Interchange Code) set.

edited data item. A data item that has been modified by suppressing zeroes and/or inserting editing characters.

*** editing character.** A single character or a fixed two-character combination belonging to the following set:

Character	Meaning
•	space
0	zero
+	plus
—	minus
CR	credit
DB	debit
Z	zero suppress
*	check protect
\$	currency sign
,	comma (decimal point)
.	period (decimal point)
/	slant (virgule, slash)

element (text element). One logical unit of a string of text, such as the description of a single data item or verb, preceded by a unique code identifying the element type.

*** elementary item.** A data item that is described as not being further logically subdivided.

enclave. When running under the Language Environment product, an enclave is analogous to a run unit. An enclave can create other enclaves on MVS and CMS by a LINK, on CMS by CMSCALL, and the use of the system () function of C.

***end class header.** A combination of words, followed by a separator period, that indicates the end of a COBOL class definition. The end class header is:
END CLASS class-name.

***end method header.** A combination of words, followed by a separator period, that indicates the end of a COBOL method definition. The end method header is:

END METHOD method-name.

*** end of Procedure Division.** The physical position of a COBOL source program after which no further procedures appear.

*** end program header.** A combination of words, followed by a separator period, that indicates the end of a COBOL source program. The end program header is:

END PROGRAM program-name.

*** entry.** Any descriptive set of consecutive clauses terminated by a separator period and written in the IDENTIFICATION DIVISION, ENVIRONMENT DIVISION, or DATA DIVISION of a COBOL program.

*** environment clause.** A clause that appears as part of an ENVIRONMENT DIVISION entry.

ENVIRONMENT DIVISION. One of the four main component parts of a COBOL program, class definition, or method definition. The ENVIRONMENT DIVISION describes the computers upon which the source program is compiled and those on which the object program is executed, and provides a linkage between the logical concept of files and their records, and the physical aspects of the devices on which files are stored.

environment-name. A name, specified by IBM, that identifies system logical units, printer and card punch control characters, report codes, and/or program switches. When an environment-name is associated with a mnemonic-name in the ENVIRONMENT DIVISION, the mnemonic-name may then be substituted in any format in which such substitution is valid.

environment variable. Any of a number of variables that describe the way an operating system is going to run and the devices it is going to recognize.

execution time. See "run time."

execution-time environment. See “run-time environment.”

* **explicit scope terminator.** A reserved word that terminates the scope of a particular Procedure Division statement.

exponent. A number, indicating the power to which another number (the base) is to be raised. Positive exponents denote multiplication, negative exponents denote division, fractional exponents denote a root of a quantity. In COBOL, an exponential expression is indicated with the symbol ‘**’ followed by the exponent.

* **expression.** An arithmetic or conditional expression.

* **extend mode.** The state of a file after execution of an OPEN statement, with the EXTEND phrase specified for that file, and before the execution of a CLOSE statement, without the REEL or UNIT phrase for that file.

extensions. Certain COBOL syntax and semantics supported by IBM compilers in addition to those described in ANSI Standard.

* **external data.** The data described in a program as external data items and external file connectors.

* **external data item.** A data item which is described as part of an external record in one or more programs of a run unit and which itself may be referenced from any program in which it is described.

* **external data record.** A logical record which is described in one or more programs of a run unit and whose constituent data items may be referenced from any program in which they are described.

external decimal item. A format for representing numbers in which the digit is contained in bits 4 through 7 and the sign is contained in bits 0 through 3 of the rightmost byte. Bits 0 through 3 of all other bytes contain 1's (hex F). For example, the decimal value of +123 is represented as 1111 0001 1111 0010 1111 0011. (Also known as “zoned decimal item.”)

* **external file connector.** A file connector which is accessible to one or more object programs in the run unit.

external floating-point item. A format for representing numbers in which a real number is represented by a pair of distinct numerals. In a floating-point representation, the real number is the product of the fixed-point part (the first numeral), and a value obtained by raising the implicit floating-point base to a power denoted by the exponent (the second numeral).

For example, a floating-point representation of the number 0.0001234 is: 0.1234 -3, where 0.1234 is the mantissa and -3 is the exponent.

* **external switch.** A hardware or software device, defined and named by the implementor, which is used to indicate that one of two alternate states exists.

F

* **figurative constant.** A compiler-generated value referenced through the use of certain reserved words.

* **file.** A collection of logical records.

* **file attribute conflict condition.** An unsuccessful attempt has been made to execute an input-output operation on a file and the file attributes, as specified for that file in the program, do not match the fixed attributes for that file.

* **file clause.** A clause that appears as part of any of the following DATA DIVISION entries: file description entry (FD entry) and sort-merge file description entry (SD entry).

* **file connector.** A storage area which contains information about a file and is used as the linkage between a file-name and a physical file and between a file-name and its associated record area.

File-Control. The name of an ENVIRONMENT DIVISION paragraph in which the data files for a given source program are declared.

* **file control entry.** A SELECT clause and all its subordinate clauses which declare the relevant physical attributes of a file.

* **file description entry.** An entry in the File Section of the DATA DIVISION that is composed of the level indicator FD, followed by a file-name, and then followed by a set of file clauses as required.

* **file-name.** A user-defined word that names a file connector described in a file description entry or a sort-merge file description entry within the File Section of the DATA DIVISION.

* **file organization.** The permanent logical file structure established at the time that a file is created.

* **file position indicator.** A conceptual entity that contains the value of the current key within the key of reference for an indexed file, or the record number of the current record for a sequential file, or the relative record number of the current record for a relative file, or indicates that no next logical record exists, or that an optional input file is not present, or that the at end condition already exists, or that no valid next record has been established.

*** File Section.** The section of the DATA DIVISION that contains file description entries and sort-merge file description entries together with their associated record descriptions.

file system. The collection of files and file management structures on a physical or logical mass storage device, such as a diskette or minidisk.

*** fixed file attributes.** Information about a file which is established when a file is created and cannot subsequently be changed during the existence of the file. These attributes include the organization of the file (sequential, relative, or indexed), the prime record key, the alternate record keys, the code set, the minimum and maximum record size, the record type (fixed or variable), the collating sequence of the keys for indexed files, the blocking factor, the padding character, and the record delimiter.

*** fixed length record.** A record associated with a file whose file description or sort-merge description entry requires that all records contain the same number of character positions.

fixed-point number. A numeric data item defined with a PICTURE clause that specifies the location of an optional sign, the number of digits it contains, and the location of an optional decimal point. The format may be either binary, packed decimal, or external decimal.

floating-point number. A numeric data item containing a fraction and an exponent. Its value is obtained by multiplying the fraction by the base of the numeric data item raised to the power specified by the exponent.

*** format.** A specific arrangement of a set of data.

*** function.** A temporary data item whose value is determined at the time the function is referenced during the execution of a statement.

*** function-identifier.** A syntactically correct combination of character-strings and separators that references a function. The data item represented by a function is uniquely identified by a function-name with its arguments, if any. A function-identifier may include a reference-modifier. A function-identifier that references an alphanumeric function may be specified anywhere in the general formats that an identifier may be specified, subject to certain restrictions. A function-identifier that references an integer or numeric function may be referenced anywhere in the general formats that an arithmetic expression may be specified.

function-name. A word that names the mechanism whose invocation, along with required arguments, determines the value of a function.

G

*** global name.** A name which is declared in only one program but which may be referenced from that program and from any program contained within that program. Condition-names, data-names, file-names, record-names, report-names, and some special registers may be global names.

*** group item.** A data item that is composed of subordinate data items.

H

header label. (1) A file label or data set label that precedes the data records on a unit of recording media. (2) Synonym for beginning-of-file label.

*** high order end.** The leftmost character of a string of characters.

I

IBM COBOL extension. Certain COBOL syntax and semantics supported by IBM compilers in addition to those described in ANSI Standard.

IDENTIFICATION DIVISION. One of the four main component parts of a COBOL program, class definition, or method definition. The IDENTIFICATION DIVISION identifies the program name, class name, or method name. The IDENTIFICATION DIVISION may include the following documentation: author name, installation, or date.

*** identifier.** A syntactically correct combination of character-strings and separators that names a data item. When referencing a data item that is not a function, an identifier consists of a data-name, together with its qualifiers, subscripts, and reference-modifier, as required for uniqueness of reference. When referencing a data item which is a function, a function-identifier is used.

IGZCBSN. The COBOL for MVS and VM bootstrap routine. It must be link-edited with any module that contains a COBOL for MVS and VM program.

*** imperative statement.** A statement that either begins with an imperative verb and specifies an unconditional action to be taken or is a conditional statement that is delimited by its explicit scope terminator (delimited scope statement). An imperative statement may consist of a sequence of imperative statements.

*** implicit scope terminator.** A separator period which terminates the scope of any preceding unterminated statement, or a phrase of a statement which by its occurrence indicates the end of the scope of any statement contained within the preceding phrase.

* **index.** A computer storage area or register, the content of which represents the identification of a particular element in a table.

* **index data item.** A data item in which the values associated with an index-name can be stored in a form specified by the implementor.

indexed data-name. An identifier that is composed of a data-name, followed by one or more index-names enclosed in parentheses.

* **indexed file.** A file with indexed organization.

* **indexed organization.** The permanent logical file structure in which each record is identified by the value of one or more keys within that record.

indexing. Synonymous with subscripting using index-names.

* **index-name.** A user-defined word that names an index associated with a specific table.

* **inheritance (for classes).** A mechanism for using the implementation of one or more *classes* as the basis for another class. A *sub-class* inherits from one or more *super-classes*. By definition the inheriting class conforms to the inherited classes.

* **initial program.** A program that is placed into an initial state every time the program is called in a run unit.

* **initial state.** The state of a program when it is first called in a run unit.

inline. In a program, instructions that are executed sequentially, without branching to routines, subroutines, or other programs.

* **input file.** A file that is opened in the INPUT mode.

* **input mode.** The state of a file after execution of an OPEN statement, with the INPUT phrase specified, for that file and before the execution of a CLOSE statement, without the REEL or UNIT phrase for that file.

* **input-output file.** A file that is opened in the I-O mode.

* **INPUT-OUTPUT SECTION.** The section of the ENVIRONMENT DIVISION that names the files and the external media required by an object program or method and that provides information required for transmission and handling of data during execution of the object program or method definition.

* **Input-Output statement.** A statement that causes files to be processed by performing operations upon individual records or upon the file as a unit. The input-output statements are: ACCEPT (with the

identifier phrase), CLOSE, DELETE, DISPLAY, OPEN, READ, REWRITE, SET (with the TO ON or TO OFF phrase), START, and WRITE.

* **input procedure.** A set of statements, to which control is given during the execution of a SORT statement, for the purpose of controlling the release of specified records to be sorted.

instance data. Data defining the state of an object. The instance data introduced by a class is defined in the WORKING-STORAGE SECTION of the DATA DIVISION of the class definition. The state of an object also includes the state of the instance variables introduced by base classes that are inherited by the current class. A separate copy of the instance data is created for each object instance.

* **integer.** (1) A numeric literal that does not include any digit positions to the right of the decimal point.

(2) A numeric data item defined in the DATA DIVISION that does not include any digit positions to the right of the decimal point.

(3) A numeric function whose definition provides that all digits to the right of the decimal point are zero in the returned value for any possible evaluation of the function.

integer function. A function whose category is numeric and whose definition does not include any digit positions to the right of the decimal point.

interface. The information that a *client* must know to use a *class*—the names of its *attributes* and the signatures of its *methods*. With direct-to-SOM compilers such as COBOL, the interface to a class may be defined by native language syntax for class definitions. Classes implemented in other languages might have their interfaces defined directly in SOM Interface Definition Language (IDL). The COBOL compiler has a compiler option, IDLGEN, to automatically generate IDL for a COBOL class.

Interface Definition Language (IDL). The formal language (independent of any programming language) by which the *interface* for a class of *objects* is defined in a IDL file, which the SOM compiler then interprets to create an implementation template file and binding files. SOM's Interface Definition Language is fully compliant with standards established by the Object Management Group's Common Object Request Broker Architecture (CORBA).

interlanguage communication (ILC). The ability of routines written in different programming languages to communicate. ILC support allows the application writer to readily build applications from component routines written in a variety of languages.

intermediate result. An intermediate field containing the results of a succession of arithmetic operations.

* **internal data.** The data described in a program excluding all external data items and external file connectors. Items described in the LINKAGE SECTION of a program are treated as internal data.

* **internal data item.** A data item which is described in one program in a run unit. An internal data item may have a global name.

internal decimal item. A format in which each byte in a field except the rightmost byte represents two numeric digits. The rightmost byte contains one digit and the sign. For example, the decimal value +123 is represented as 0001 0010 0011 1111. (Also known as packed decimal.)

* **internal file connector.** A file connector which is accessible to only one object program in the run unit.

* **intra-record data structure.** The entire collection of groups and elementary data items from a logical record which is defined by a contiguous subset of the data description entries which describe that record. These data description entries include all entries whose level-number is greater than the level-number of the first data description entry describing the intra-record data structure.

intrinsic function. A pre-defined function, such as a commonly used arithmetic function, called by a built-in function reference.

* **invalid key condition.** A condition, at object time, caused when a specific value of the key associated with an indexed or relative file is determined to be invalid.

* **I-O-CONTROL.** The name of an ENVIRONMENT DIVISION paragraph in which object program requirements for rerun points, sharing of same areas by several data files, and multiple file storage on a single input-output device are specified.

* **I-O-CONTROL entry.** An entry in the I-O-CONTROL paragraph of the ENVIRONMENT DIVISION which contains clauses that provide information required for the transmission and handling of data on named files during the execution of a program.

* **I-O-Mode.** The state of a file after execution of an OPEN statement, with the I-O phrase specified, for that file and before the execution of a CLOSE statement without the REEL or UNIT phase for that file.

* **I-O status.** A conceptual entity which contains the two-character value indicating the resulting status of an input-output operation. This value is made available to the program through the use of the FILE STATUS clause in the file control entry for the file.

iteration structure. A program processing logic in which a series of statements is repeated while a condition is true or until a condition is true.

K

K. When referring to storage capacity, two to the tenth power; 1024 in decimal notation.

* **key.** A data item that identifies the location of a record, or a set of data items which serve to identify the ordering of data.

* **key of reference.** The key, either prime or alternate, currently being used to access records within an indexed file.

* **key word.** A reserved word or function-name whose presence is required when the format in which the word appears is used in a source program.

kilobyte (KB). One kilobyte equals 1024 bytes.

L

* **language-name.** A system-name that specifies a particular programming language.

Language Environment-conforming. A characteristic of compiler products (COBOL/370, COBOL for MVS and VM, AD/Cycle C/370, C/C++ for MVS and VM, PL/I for MVS and VM) that produce object code conforming to the Language Environment format.

last-used state. A program is in last-used state if its internal values remain the same as when the program was exited (are not reset to their initial values).

* **letter.** A character belonging to one of the following two sets:

1. Uppercase letters: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z
2. Lowercase letters: a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z

* **level indicator.** Two alphabetic characters that identify a specific type of file or a position in a hierarchy. The level indicators in the DATA DIVISION are: CD, FD, and SD.

* **level-number.** A user-defined word, expressed as a two digit number, which indicates the hierarchical position of a data item or the special properties of a data description entry. Level-numbers in the range from 1 through 49 indicate the position of a data item in the hierarchical structure of a logical record. Level-numbers in the range 1 through 9 may be written either as a single digit or as a zero followed by a significant digit. Level-numbers 66, 77 and 88 identify special properties of a data description entry.

* **library-name.** A user-defined word that names a COBOL library that is to be used by the compiler for a given source program compilation.

* **library text.** A sequence of text words, comment lines, the separator space, or the separator pseudo-text delimiter in a COBOL library.

LILIAN DATE. The number of days since the beginning of the Gregorian calendar. Day one is Friday, October 15, 1582. The Lilian date format is named in honor of Luigi Lilio, the creator of the Gregorian calendar.

* **LINAGE-COUNTER.** A special register whose value points to the current position within the page body.

LINKAGE SECTION. The section in the DATA DIVISION of the called program that describes data items available from the calling program. These data items may be referred to by both the calling and called program.

literal. A character-string whose value is specified either by the ordered set of characters comprising the string, or by the use of a figurative constant.

local. A set of attributes for a program execution environment indicating culturally sensitive considerations, such as: character code page, collating sequence, date/time format, monetary value representation, numeric value representation, or language.

* **LOCAL-STORAGE SECTION.** The section of the DATA DIVISION that defines storage that is allocated and freed on a per-invocation basis, depending on the value assigned in their VALUE clauses.

* **logical operator.** One of the reserved words AND, OR, or NOT. In the formation of a condition, either AND, or OR, or both can be used as logical connectives. NOT can be used for logical negation.

* **logical record.** The most inclusive data item. The level-number for a record is 01. A record may be either an elementary item or a group of items. The term is synonymous with record.

* **low order end.** The rightmost character of a string of characters.

M

main program. In a hierarchy of programs and subroutines, the first program to receive control when the programs are run.

* **mass storage.** A storage medium in which data may be organized and maintained in both a sequential and nonsequential manner.

* **mass storage device.** A device having a large storage capacity; for example, magnetic disk, magnetic drum.

* **mass storage file.** A collection of records that is assigned to a mass storage medium.

* **megabyte (M).** One megabyte equals 1,048,576 bytes.

* **merge file.** A collection of records to be merged by a MERGE statement. The merge file is created and can be used only by the merge function.

metaclass. A SOM class whose instances are SOM class-objects. The methods defined in metaclasses are executed without requiring any object instances of the class to exist, and are frequently used to create instances of the class.

method. Procedural code that defines one of the operations supported by an object, and that is executed by an INVOKE statement on that object.

* **Method Definition.** The COBOL source unit that defines a method.

* **method identification entry.** An entry in the METHOD-ID paragraph of the IDENTIFICATION DIVISION which contains clauses that specify the method-name and assign selected attributes to the method definition.

* **method-name.** A user-defined word that identifies a method.

* **mnemonic-name.** A user-defined word that is associated in the ENVIRONMENT DIVISION with a specified implementor-name.

multitasking. Mode of operation that provides for the concurrent, or interleaved, execution of two or more tasks. When running under the Language Environment product, multitasking is synonymous with *multithreading*.

MVS/XA* (Multiple Virtual Storage/Extended Architecture). An IBM operating system that manages multiple virtual address spaces in IBM processors operating in extended architecture mode. MVS/XA supports the 31-bit addressing mechanism of extended architecture mode, and therefore, allows an address space as large as 2³¹ bytes (2048 megabytes or 2 gigabytes).

N

name. A word composed of not more than 30 characters that defines a COBOL operand.

* **native character set.** The implementor-defined character set associated with the computer specified in the OBJECT-COMPUTER paragraph.

* **native collating sequence.** The implementor-defined collating sequence associated with the computer specified in the OBJECT-COMPUTER paragraph.

* **negated combined condition.** The 'NOT' logical operator immediately followed by a parenthesized combined condition.

* **negated simple condition.** The 'NOT' logical operator immediately followed by a simple condition.

nested program. A program that is directly contained within another program.

* **next executable sentence.** The next sentence to which control will be transferred after execution of the current statement is complete.

* **next executable statement.** The next statement to which control will be transferred after execution of the current statement is complete.

* **next record.** The record that logically follows the current record of a file.

* **noncontiguous items.** Elementary data items in the WORKING-STORAGE and LINKAGE SECTIONS that bear no hierarchic relationship to other data items.

* **nonnumeric item.** A data item whose description permits its content to be composed of any combination of characters taken from the computer's character set. Certain categories of nonnumeric items may be formed from more restricted character sets.

* **nonnumeric literal.** A literal bounded by quotation marks. The string of characters may include any character in the computer's character set.

null. Figurative constant used to assign the value of an invalid address to pointer data items. NULLS can be used wherever NULL can be used.

* **numeric character.** A character that belongs to the following set of digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

numeric-edited item. A numeric item that is in such a form that it may be used in printed output. It may consist of external decimal digits from 0 through 9, the decimal point, commas, the dollar sign, editing sign control symbols, plus other editing symbols.

* **numeric function.** A function whose class and category are numeric but which for some possible evaluation does not satisfy the requirements of integer functions.

* **numeric item.** A data item whose description restricts its content to a value represented by characters chosen from the digits from '0' through '9'; if signed, the item may also contain a '+', '-', or other representation of an operational sign.

* **numeric literal.** A literal composed of one or more numeric characters that may contain either a decimal point, or an algebraic sign, or both. The decimal point must not be the rightmost character. The algebraic sign, if present, must be the leftmost character.

O

object. An entity that has state (its data values) and operations (its methods). An object is a way to encapsulate state and behavior.

object code. Output from a compiler or assembler that is itself executable machine code or is suitable for processing to produce executable machine code.

* **OBJECT-COMPUTER.** The name of an ENVIRONMENT DIVISION paragraph in which the computer environment, within which the object program is executed, is described.

* **object computer entry.** An entry in the OBJECT-COMPUTER paragraph of the ENVIRONMENT DIVISION which contains clauses that describe the computer environment in which the object program is to be executed.

object deck. A portion of an object program suitable as input to a linkage editor. Synonymous with *object module* and *text deck*.

object module. Synonym for *object deck* or *text deck*.

* **object of entry.** A set of operands and reserved words, within a DATA DIVISION entry of a COBOL program, that immediately follows the subject of the entry.

* **object program.** A set or group of executable machine language instructions and other material designed to interact with data to provide problem solutions. In this context, an object program is generally the machine language result of the operation of a COBOL compiler on a source program. Where there is no danger of ambiguity, the word 'program' alone may be used in place of the phrase 'object program.'

* **object time.** The time at which an object program is executed. The term is synonymous with execution time.

* **obsolete element.** A COBOL language element in Standard COBOL that is to be deleted from the next revision of Standard COBOL.

ODO object. In the example below,

```
WORKING-STORAGE SECTION
01  TABLE-1.
    05  X                                PICS9.
    05  Y OCCURS 3 TIMES
        DEPENDING ON X                PIC X.
```


X is the object of the OCCURS DEPENDING ON clause (ODO object). The value of the ODO object determines how many of the ODO subject appear in the table.

ODO subject. In the example above, Y is the subject of the OCCURS DEPENDING ON clause (ODO subject). The number of Y ODO subjects that appear in the table depends on the value of X.

*** open mode.** The state of a file after execution of an OPEN statement for that file and before the execution of a CLOSE statement without the REEL or UNIT phrase for that file. The particular open mode is specified in the OPEN statement as either INPUT, OUTPUT, I-O or EXTEND.

*** operand.** Whereas the general definition of operand is "that component which is operated upon," for the purposes of this document, any lowercase word (or words) that appears in a statement or entry format may be considered to be an operand and, as such, is an implied reference to the data indicated by the operand.

*** operational sign.** An algebraic sign, associated with a numeric data item or a numeric literal, to indicate whether its value is positive or negative.

*** optional file.** A file which is declared as being not necessarily present each time the object program is executed. The object program causes an interrogation for the presence or absence of the file.

*** optional word.** A reserved word that is included in a specific format only to improve the readability of the language and whose presence is optional to the user when the format in which the word appears is used in a source program.

OS/2 (Operating System/2*). A multi-tasking operating system for the IBM Personal Computer family that allows you to run both DOS mode and OS/2 mode programs.

*** output file.** A file that is opened in either the OUTPUT mode or EXTEND mode.

*** output mode.** The state of a file after execution of an OPEN statement, with the OUTPUT or EXTEND phrase specified, for that file and before the execution of a CLOSE statement without the REEL or UNIT phrase for that file.

*** output procedure.** A set of statements to which control is given during execution of a SORT statement after the sort function is completed, or during execution of a MERGE statement after the merge function reaches a point at which it can select the next record in merged order when requested.

overflow condition. A condition that occurs when a portion of the result of an operation exceeds the capacity of the intended unit of storage.

P

packed decimal item. See "internal decimal item."

*** padding character.** An alphanumeric character used to fill the unused character positions in a physical record.

page. A vertical division of output data representing a physical separation of such data, the separation being based on internal logical requirements and/or external characteristics of the output medium.

*** page body.** That part of the logical page in which lines can be written and/or spaced.

*** paragraph.** In the Procedure Division, a paragraph-name followed by a separator period and by zero, one, or more sentences. In the IDENTIFICATION and ENVIRONMENT DIVISIONs, a paragraph header followed by zero, one, or more entries.

*** paragraph header.** A reserved word, followed by the separator period, that indicates the beginning of a paragraph in the IDENTIFICATION and ENVIRONMENT DIVISIONs. The permissible paragraph headers in the IDENTIFICATION DIVISION are:

PROGRAM-ID. (Program IDENTIFICATION DIVISION only)
CLASS-ID. (Class IDENTIFICATION DIVISION only)
METHOD-ID. (Method IDENTIFICATION DIVISION only)
AUTHOR.
INSTALLATION.
DATE-WRITTEN.
DATE-COMPILED.
SECURITY.

The permissible paragraph headers in the ENVIRONMENT DIVISION are:

SOURCE-COMPUTER.
OBJECT-COMPUTER.
SPECIAL-NAMES.
REPOSITORY. (Program or Class CONFIGURATION SECTION only)
FILE-CONTROL.
I-O-CONTROL.

*** paragraph-name.** A user-defined word that identifies and begins a paragraph in the Procedure Division.

parameter. Parameters are used to pass data values between calling and called programs.

password. A unique string of characters that a program, computer operator, or user must supply to meet security requirements before gaining access to data.

* **phrase.** A phrase is an ordered set of one or more consecutive COBOL character-strings that form a portion of a COBOL procedural statement or of a COBOL clause.

* **physical record.** See "block."

pointer data item. A data item in which address values can be stored. Data items are explicitly defined as pointers with the USAGE IS POINTER clause. ADDRESS OF special registers are implicitly defined as pointer data items. Pointer data items can be compared for equality or moved to other pointer data items.

portability. The ability to transfer an application program from one application platform to another with relatively few changes to the source program.

preloaded. In COBOL this refers to COBOL programs that remain resident in storage under IMS instead of being loaded each time they are called.

* **prime record key.** A key whose contents uniquely identify a record within an indexed file.

* **priority-number.** A user-defined word which classifies sections in the Procedure Division for purposes of segmentation. Segment-numbers may contain only the characters '0','1', ... , '9'. A segment-number may be expressed either as a one- or two-digit number.

* **procedure.** A paragraph or group of logically successive paragraphs, or a section or group of logically successive sections, within the Procedure Division.

* **procedure branching statement.** A statement that causes the explicit transfer of control to a statement other than the next executable statement in the sequence in which the statements are written in the source program. The procedure branching statements are: ALTER, CALL, EXIT, EXIT PROGRAM, GO TO, MERGE, (with the OUTPUT PROCEDURE phrase), PERFORM and SORT (with the INPUT PROCEDURE or OUTPUT PROCEDURE phrase).

Procedure Division. One of the four main component parts of a COBOL program, class definition, or method definition. The Procedure Division contains instructions for solving a problem. The Program and Method Procedure Divisions may contain imperative statements, conditional statements, compiler directing statements, paragraphs, procedures, and sections. The Class Procedure Division contains only method definitions.

procedure integration. One of the functions of the COBOL optimizer is to simplify calls to performed procedures or contained programs.

PERFORM procedure integration is the process whereby a PERFORM statement is replaced by its

performed procedures. Contained program procedure integration is the process where a CALL to a contained program is replaced by the program code.

* **procedure-name.** A user-defined word that is used to name a paragraph or section in the Procedure Division. It consists of a paragraph-name (which may be qualified) or a section-name.

procedure-pointer data item. A data item in which a pointer to an entry point can be stored. A data item defined with the USAGE IS PROCEDURE-POINTER clause contains the address of a procedure entry point.

* **program identification entry.** An entry in the PROGRAM-ID paragraph of the IDENTIFICATION DIVISION which contains clauses that specify the program-name and assign selected program attributes to the program.

* **program-name.** In the IDENTIFICATION DIVISION and the end program header, a user-defined word that identifies a COBOL source program.

* **pseudo-text.** A sequence of text words, comment lines, or the separator space in a source program or COBOL library bounded by, but not including, pseudo-text delimiters.

* **pseudo-text delimiter.** Two contiguous equal sign characters (==) used to delimit pseudo-text.

* **punctuation character.** A character that belongs to the following set:

Character	Meaning
,	comma
;	semicolon
:	colon
.	period (full stop)
"	quotation mark
(	left parenthesis
)	right parenthesis
•	space
=	equal sign

Q

QSAM (Queued Sequential Access Method). An extended version of the basic sequential access method (BSAM). When this method is used, a queue is formed of input data blocks that are awaiting processing or of output data blocks that have been processed and are awaiting transfer to auxiliary storage or to an output device.

* **qualified data-name.** An identifier that is composed of a data-name followed by one or more sets of either of the connectives OF and IN followed by a data-name qualifier.

* **qualifier.**

1. A data-name or a name associated with a level indicator which is used in a reference either together with another data-name which is the name of an item that is subordinate to the qualifier or together with a condition-name.
2. A section-name that is used in a reference together with a paragraph-name specified in that section.
3. A library-name that is used in a reference together with a text-name associated with that library.

R

*** random access.** An access mode in which the program-specified value of a key data item identifies the logical record that is obtained from, deleted from, or placed into a relative or indexed file.

*** record.** See "logical record."

*** record area.** A storage area allocated for the purpose of processing the record described in a record description entry in the File Section of the DATA DIVISION. In the File Section, the current number of character positions in the record area is determined by the explicit or implicit RECORD clause.

*** record description.** See "record description entry."

*** record description entry.** The total set of data description entries associated with a particular record. The term is synonymous with record description.

recording mode. The format of the logical records in a file. Recording mode can be F (fixed-length), V (variable-length), S (spanned), or U (undefined).

record key. A key whose contents identify a record within an indexed file.

*** record-name.** A user-defined word that names a record described in a record description entry in the DATA DIVISION of a COBOL program.

*** record number.** The ordinal number of a record in the file whose organization is sequential.

recursion. A program calling itself or being directly or indirectly called by a one of its called programs.

recursively capable. A program is recursively capable (can be called recursively) if the RECURSIVE attribute is on the PROGRAM-ID statement.

reel. A discrete portion of a storage medium, the dimensions of which are determined by each implementor that contains part of a file, all of a file, or any number of files. The term is synonymous with unit and volume.

reentrant. The attribute of a program or routine that allows more than one user to share a single copy of a load module.

*** reference format.** A format that provides a standard method for describing COBOL source programs.

reference modification. A method of defining a new alphanumeric data item by specifying the leftmost character and length relative to the leftmost character of another alphanumeric data item.

*** reference-modifier.** A syntactically correct combination of character-strings and separators that defines a unique data item. It includes a delimiting left parenthesis separator, the leftmost character position, a colon separator, optionally a length, and a delimiting right parenthesis separator.

*** relation.** See "relational operator" or "relation condition."

*** relational operator.** A reserved word, a relation character, a group of consecutive reserved words, or a group of consecutive reserved words and relation characters used in the construction of a relation condition. The permissible operators and their meanings are:

Operator	Meaning
IS GREATER THAN	Greater than
IS >	Greater than
IS NOT GREATER THAN	Not greater than
IS NOT >	Not greater than
IS LESS THAN	Less than
IS <	Less than
IS NOT LESS THAN	Not less than
IS NOT <	Not less than
IS EQUAL TO	Equal to
IS =	Equal to
IS NOT EQUAL TO	Not equal to
IS NOT =	Not equal to
IS GREATER THAN OR EQUAL TO	Greater than or equal to
IS >=	Greater than or equal to
IS LESS THAN OR EQUAL TO	Less than or equal to
IS <=	Less than or equal to

*** relation character.** A character that belongs to the following set:

Character	Meaning
>	greater than
<	less than
=	equal to

* **relation condition.** The proposition, for which a truth value can be determined, that the value of an arithmetic expression, data item, nonnumeric literal, or index-name has a specific relationship to the value of another arithmetic expression, data item, nonnumeric literal, or index name. (See also "relational operator.")

* **relative file.** A file with relative organization.

* **relative key.** A key whose contents identify a logical record in a relative file.

* **relative organization.** The permanent logical file structure in which each record is uniquely identified by an integer value greater than zero, which specifies the record's logical ordinal position in the file.

* **relative record number.** The ordinal number of a record in a file whose organization is relative. This number is treated as a numeric literal which is an integer.

* **reserved word.** A COBOL word specified in the list of words that may be used in a COBOL source program, but that must not appear in the program as user-defined words or system-names.

* **resource.** A facility or service, controlled by the operating system, that can be used by an executing program.

* **resultant identifier.** A user-defined data item that is to contain the result of an arithmetic operation.

reusable environment. A reusable environment is when you establish an assembler program as the main program by using either ILBOSTP0 programs, IGZERRE programs, or the RTEREUS run-time option.

routine. A set of statements in a COBOL program that causes the computer to perform an operation or series of related operations. In Language Environment, refers to either a procedure, function, or subroutine.

* **routine-name.** A user-defined word that identifies a procedure written in a language other than COBOL.

* **run time.** The time at which an object program is executed. The term is synonymous with object time.

run-time environment. The environment in which a COBOL program executes.

* **run unit.** A stand-alone object program, or several object programs, that interact via COBOL CALL statements, which function at run time as an entity.

S

SBCS (Single Byte Character Set). See "Single Byte Character Set (SBCS)".

scope terminator. A COBOL reserved word that marks the end of certain Procedure Division statements. It may be either explicit (END-ADD, for example) or implicit (separator period).

* **section.** A set of zero, one or more paragraphs or entities, called a section body, the first of which is preceded by a section header. Each section consists of the section header and the related section body.

* **section header.** A combination of words followed by a separator period that indicates the beginning of a section in the Environment, Data, and Procedure Divisions. In the ENVIRONMENT and DATA DIVISIONs, a section header is composed of reserved words followed by a separator period. The permissible section headers in the ENVIRONMENT DIVISION are:

CONFIGURATION SECTION.
INPUT-OUTPUT SECTION.

The permissible section headers in the DATA DIVISION are:

FILE SECTION.
WORKING-STORAGE SECTION.
LOCAL-STORAGE SECTION.
LINKAGE SECTION.

In the Procedure Division, a section header is composed of a section-name, followed by the reserved word SECTION, followed by a separator period.

* **section-name.** A user-defined word that names a section in the Procedure Division.

selection structure. A program processing logic in which one or another series of statements is executed, depending on whether a condition is true or false.

* **sentence.** A sequence of one or more statements, the last of which is terminated by a separator period.

* **separately compiled program.** A program which, together with its contained programs, is compiled separately from all other programs.

* **separator.** A character or two contiguous characters used to delimit character-strings.

* **separator comma.** A comma (,) followed by a space used to delimit character-strings.

* **separator period.** A period (.) followed by a space used to delimit character-strings.

* **separator semicolon.** A semicolon (;) followed by a space used to delimit character-strings.

sequence structure. A program processing logic in which a series of statements is executed in sequential order.

* **sequential access.** An access mode in which logical records are obtained from or placed into a file in a consecutive predecessor-to-successor logical record sequence determined by the order of records in the file.

* **sequential file.** A file with sequential organization.

* **sequential organization.** The permanent logical file structure in which a record is identified by a predecessor-successor relationship established when the record is placed into the file.

serial search. A search in which the members of a set are consecutively examined, beginning with the first member and ending with the last.

* **77-level-description-entry.** A data description entry that describes a noncontiguous data item with the level-number 77.

* **sign condition.** The proposition, for which a truth value can be determined, that the algebraic value of a data item or an arithmetic expression is either less than, greater than, or equal to zero.

* **simple condition.** Any single condition chosen from the set:

- Relation condition
- Class condition
- Condition-name condition
- Switch-status condition
- Sign condition

Single Byte Character Set (SBCS). A set of characters in which each character is represented by a single byte. See also "EBCDIC (Extended Binary-Coded Decimal Interchange Code)."

slack bytes. Bytes inserted between data items or records to ensure correct alignment of some numeric items. Slack bytes contain no meaningful data. In some cases, they are inserted by the compiler; in others, it is the responsibility of the programmer to insert them. The SYNCHRONIZED clause instructs the compiler to insert slack bytes when they are needed for proper alignment. Slack bytes between records are inserted by the programmer.

SOM. *System Object Model*

* **sort file.** A collection of records to be sorted by a SORT statement. The sort file is created and can be used by the sort function only.

* **sort-merge file description entry.** An entry in the File Section of the DATA DIVISION that is composed of the level indicator SD, followed by a file-name, and then followed by a set of file clauses as required.

* **SOURCE-COMPUTER.** The name of an ENVIRONMENT DIVISION paragraph in which the computer environment, within which the source program is compiled, is described.

* **source computer entry.** An entry in the SOURCE-COMPUTER paragraph of the ENVIRONMENT DIVISION which contains clauses that describe the computer environment in which the source program is to be compiled.

* **source item.** An identifier designated by a SOURCE clause that provides the value of a printable item.

source program. Although it is recognized that a source program may be represented by other forms and symbols, in this document it always refers to a syntactically correct set of COBOL statements. A COBOL source program commences with the IDENTIFICATION DIVISION or a COPY statement. A COBOL source program is terminated by the end program header, if specified, or by the absence of additional source program lines.

* **special character.** A character that belongs to the following set:

Character	Meaning
+	plus sign
-	minus sign (hyphen)
*	asterisk
/	slant (virgule, slash)
=	equal sign
\$	currency sign
,	comma (decimal point)
;	semicolon
.	period (decimal point, full stop)
"	quotation mark
(	left parenthesis
)	right parenthesis
>	greater than symbol
<	less than symbol
:	colon

* **special-character word.** A reserved word that is an arithmetic operator or a relation character.

SPECIAL-NAMES. The name of an ENVIRONMENT DIVISION paragraph in which environment-names are related to user-specified mnemonic-names.

* **special names entry.** An entry in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION which provides means for specifying the currency sign; choosing the decimal point; specifying symbolic characters; relating implementor-names to user-specified mnemonic-names; relating alphabet-names to character sets or collating

sequences; and relating class-names to sets of characters.

* **special registers.** Certain compiler generated storage areas whose primary use is to store information produced in conjunction with the use of a specific COBOL feature.

* **standard data format.** The concept used in describing the characteristics of data in a COBOL DATA DIVISION under which the characteristics or properties of the data are expressed in a form oriented to the appearance of the data on a printed page of infinite length and breadth, rather than a form oriented to the manner in which the data is stored internally in the computer, or on a particular external medium.

* **statement.** A syntactically valid combination of words, literals, and separators, beginning with a verb, written in a COBOL source program.

structured programming. A technique for organizing and coding a computer program in which the program comprises a hierarchy of segments, each segment having a single entry point and a single exit point. Control is passed downward through the structure without unconditional branches to higher levels of the hierarchy.

* **sub-class.** A class that inherits from another class. When two classes in an inheritance relationship are considered together, the sub-class is the inheritor or inheriting class; the *super-class* is the inheritee or inherited class.

* **subject of entry.** An operand or reserved word that appears immediately following the level indicator or the level-number in a DATA DIVISION entry.

* **subprogram.** See "called program."

* **subscript.** An occurrence number represented by either an integer, a data-name optionally followed by an integer with the operator + or -, or an index-name optionally followed by an integer with the operator + or -, that identifies a particular element in a table. A subscript may be the word ALL when the subscripted identifier is used as a function argument for a function allowing a variable number of arguments.

* **subscripted data-name.** An identifier that is composed of a data-name followed by one or more subscripts enclosed in parentheses.

* **super-class.** A class that is inherited by another class. See also *sub-class*.

switch-status condition. The proposition, for which a truth value can be determined, that an UPSI switch, capable of being set to an 'on' or 'off' status, has been set to a specific status.

* **symbolic-character.** A user-defined word that specifies a user-defined figurative constant.

syntax. (1) The relationship among characters or groups of characters, independent of their meanings or the manner of their interpretation and use. (2) The structure of expressions in a language. (3) The rules governing the structure of a language. (4) The relationship among symbols. (5) The rules for the construction of a statement.

* **system-name.** A COBOL word that is used to communicate with the operating environment.

System Object Model (SOM). IBM's object-oriented programming technology for building, packaging, and manipulating class libraries. SOM conforms to the Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA) standards.

T

* **table.** A set of logically consecutive items of data that are defined in the DATA DIVISION by means of the OCCURS clause.

* **table element.** A data item that belongs to the set of repeated items comprising a table.

text deck. Synonym for *object deck* or *object module*.

* **text-name.** A user-defined word that identifies library text.

* **text word.** A character or a sequence of contiguous characters between margin A and margin R in a COBOL library, source program, or in pseudo-text which is:

- A separator, except for: space; a pseudo-text delimiter; and the opening and closing delimiters for nonnumeric literals. The right parenthesis and left parenthesis characters, regardless of context within the library, source program, or pseudo-text, are always considered text words.
- A literal including, in the case of nonnumeric literals, the opening quotation mark and the closing quotation mark that bound the literal.
- Any other sequence of contiguous COBOL characters except comment lines and the word 'COPY' bounded by separators that are neither a separator nor a literal.

top-down design. The design of a computer program using a hierarchic structure in which related functions are performed at each level of the structure.

top-down development. See "structured programming."

trailer-label. (1) A file or data set label that follows the data records on a unit of recording medium. (2) Synonym for end-of-file label.

* **truth value.** The representation of the result of the evaluation of a condition in terms of one of two values: true or false.

U

* **unary operator.** A plus (+) or a minus (-) sign, that precedes a variable or a left parenthesis in an arithmetic expression and that has the effect of multiplying the expression by +1 or -1, respectively.

unit. A module of direct access, the dimensions of which are determined by IBM.

universal object reference. A data-name that can refer to an object of any class.

* **unsuccessful execution.** The attempted execution of a statement that does not result in the execution of all the operations specified by that statement. The unsuccessful execution of a statement does not affect any data referenced by that statement, but may affect status indicators.

UPSI switch. A program switch that performs the functions of a hardware switch. Eight are provided: UPSI-0 through UPSI-7.

* **user-defined word.** A COBOL word that must be supplied by the user to satisfy the format of a clause or statement.

V

* **variable.** A data item whose value may be changed by execution of the object program. A variable used in an arithmetic expression must be a numeric elementary item.

* **variable length record.** A record associated with a file whose file description or sort-merge description entry permits records to contain a varying number of character positions.

* **variable occurrence data item.** A variable occurrence data item is a table element which is repeated a variable number of times. Such an item must contain an OCCURS DEPENDING ON clause in its data description entry, or be subordinate to such an item.

* **variably located group.** A group item following, and not subordinate to, a variable-length table in the same level-01 record.

* **variably located item.** A data item following, and not subordinate to, a variable-length table in the same level-01 record.

* **verb.** A word that expresses an action to be taken by a COBOL compiler or object program.

VM/SP (Virtual Machine/System Product). An IBM-licensed program that manages the resources of a single computer so that multiple computing systems appear to exist. Each virtual machine is the functional equivalent of a "real" machine.

volume. A module of external storage. For tape devices it is a reel; for direct-access devices it is a unit.

volume switch procedures. System specific procedures executed automatically when the end of a unit or reel has been reached before end-of-file has been reached.

W

* **word.** A character-string of not more than 30 characters which forms a user-defined word, a system-name, a reserved word, or a function-name.

* **WORKING-STORAGE SECTION.** The section of the DATA DIVISION that describes working storage data items, composed either of noncontiguous items or working storage records or of both.

Z

zoned decimal item. See "external decimal item."

List of Abbreviations

<i>ANSI</i>	American National Standards Institute	<i>MF</i>	Micro Focus
<i>DB2</i>	Data Base 2 from IBM	<i>PM</i>	presentation manager
<i>DS</i>	Dialog System	<i>UI</i>	user interface
<i>GUI</i>	graphical user interface	<i>VDE</i>	visual development environment
<i>IBM</i>	International Business Machines Corporation	<i>VSAM</i>	virtual storage access method
<i>ITSO</i>	International Technical Support Organization		

Index

A

abbreviations 87
acronyms 87

B

bibliography 61

D

dialog system
 comparison to VDE 4
 data migration 7
 export file 17, 45
 migration approach 5
 objects 18
 sample export file 19
 structure 1, 2

M

migration utility
 description 17
 hardware requirements 57
 installation 57
 main window 17, 21
 software requirements 57
 using 21

O

objects in dialog system
 bitmaps 32
 dialog boxes 32
 entry fields 27
 group boxes 32
 list boxes 30
 menu items 26
 message boxes 33
 multiline entry fields 31
 notebooks 32
 push buttons 29
 radio button 31
 selection or combination boxes 31
 static text 31
 windows 25

P

parts in visual development environment
 bitmaps 32
 dialog boxes 32
 entry fields 27
 group boxes 32
 list boxes 30

parts in visual development environment (*continued*)

 menu items 26
 message boxes 33
 multiline entry fields 31
 notebooks 32
 push buttons 29
 radio button 31
 selection or combination boxes 31
 static text 31
 windows 25

PM application
 structure 3

R

REXX
 VX-REXX 17

S

sample application
 customer information window 13
 customer list window 13
 embedding logic code 42
 entry field event code migration 48, 51
 event logic migration 45
 main window 11
 migration 35
 migration LOG file 36
 overview 11
 push button event code migration 53, 54
 radio button event code migration 54, 55
 select model window 14
 select part number window 14
 shipid window 15
 VDE code assistant 46
 window structure 13

V

visual development environment
 comparison to dialog system 4
 data migration 7
 log file format 20
 migration approach 5
 ODG file format 20
 parts 18
 user-defined part 22



Printed in U.S.A.

S624-4887-00

